Jül-4360

Institut für Energie- und Klimaforschung
Plasmaphysik (IEK-4)

Bewertung und Implementierung von optimierten Raytracing-Algorithmen in BREP-basierten Monte Carlo Transport Codes

O. Schmidt, D. Reiter, P. Börner

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

Bewertung und Implementierung von optimierten Raytracing-Algorithmen in BREP-basierten Monte Carlo Transport Codes

O. Schmidt[1]*, D. Reiter, P. Börner*

[1] basierend auf einer Masterarbeit angefertigt von O. Schmidt an der FH Aachen, Campus Jülich, Labor für Medizinische Informatik, Prof. Dr. W. Hillen, in Zusammenarbeit mit dem IEK-4, Forschungszentrum Jülich GmbH

Berichte des Forschungszentrums Jülich; 4360
ISSN 0944-2952
Institut für Energie- und Klimaforschung
Plasmaphysik (IEK-4)
Jül-4360

Vollständig frei verfügbar im Internet auf dem Jülicher Open Access Server (JUWEL)
unter http://www.fz-juelich.de/zb/juwel

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek, Verlag
D-52425 Jülich · Bundesrepublik Deutschland
☎ 02461 61-5220 · Telefax: 02461 61-6103 · e-mail: zb-publikation@fz-juelich.de

Abstract

The Monte Carlo Transport Code EIRENE[2] has been optimized by implementation of ray-tracing algorithms adopted from computer graphics applications. In particular the original boundary representation (BREP) procedure used in EIRENE for specifying complex 3D configurations by a number of first or second order algebraic surface segments has been speeded up considerably for model cases in which the number of surface segments in the model is large. This is achieved by an Octree based optimization procedure, replacing the original linear search algorithm of EIRENE to identify the point of intersection of a straight line (the flight path of a Monte Carlo trajectory) with any of the surface segments used to specify the geometrical model of the particle transport simulation. Here we utilize the mathematical analogy between ray-tracing problems and Monte Carlo particle tracing. After successful implementation of the optimized procedure in EIRENE module `TIMEA.F` and a number of code profiling studies this module turned out not to be the most time consuming part any more; the evaluation of the statistical error estimates (standard deviations) of surface averaged quantities (e.g. fluxes, energy fluxes, sputter yields) had instead moved up in the list of most CPU critical components. Also this part of the code was then optimized by implementing the same indirect addressing concept for surface tally statistics as was already implemented in EIRENE back in the eighties of the last century for all volume averaged quantities, leading to a further significant speed-up of EIRENE for model cases of complex (many thousand surface segments) 3D configurations.

[2] www.eirene.de/manual

Zusammenfassung

Der vorliegende Report beschreibt die Beschleunigung der Schnittpunkt - Berechnung von Teilchenflugbahnen mit Geometrieflächen im Neutralteilchen-Monte-Carlo-Transport-Code (MCTC) EIRENE[3].

Der Report ist hervor gegangen aus (und ist im letzten Kapitel eine Erweiterung von) einer Masterarbeit (Technomathematik) eines der Autoren (O. Schmidt, März 2012), angefertigt an der FH Aachen. Abt. Jülich, Labor für Medizinische Informatik, Prof. Dr. W. Hillen.

Die hier beschriebene Optimierung von Monte Carlo Transport Codes basiert auf Konzepten der Bildverarbeitung (Ray-Tracing) und ist vermutlich grundsätzlich vorteilhaft in Monte Carlo Transportsimulationen einsetzbar, auch über die hier diskutierten Anwendungen im EIRENE Code hinaus.

Im EIRENE Code, einem im Forschungszentrum Jülich (FZJ) entwickelten und angewendetet Monte Carlo Teilchentransportcode, können Modellgeometrien unter anderem durch sog. „Additional Surfaces“ als ein „Boundary Representation“ (BREP) Modell definiert werden. Die Beschreibung von Umrandungen und auch inneren Flächen erfolgt mittels algebraischer Flächen 1. und 2. Ordnung. Während im EIRENE Code in den frühen achtziger Jahren nur wenige (10 - 100) solcher Flächen zum Aufbau einer vollständigen 3D Modellgeometrie verwendet wurden, sind dies inzwischen zunehmend mehr, bei automatischer Generierung mittels CAD Systemen bis zu einigen 10 000 oder sogar 100 000 solcher „additional surfaces“ geworden.

Eine Erweiterung/Optimierung des entsprechenden Moduls des EIRENE Codes zur Schnittpunktberechnung von Flugbahnen mit diesen Flächen (`TIMEA.F`) ist deshalb notwendig geworden, um die Laufzeiten bei der Monte Carlo Simulation solcher großen (detaillierten) Geometrien klein zu halten.

Inzwischen werden, insbesondere bei CAD generierten Modellgeometrien, alle Bauteile aus sehr vielen Dreiecks-Elementen im $\mathbb{R}^3$ aufgebaut, d.h. es werden in der Regel nur noch lineare algebraische Flächen verwendet. Der Grund dafür liegt darin, dass mit 3D-CAD-Software entworfene Flächenstücke nicht auf einfache Weise direkt in Monte Carlo Transport Code Geometrie-Eingaben überführt werden können, da CAD Software typisch algebraische Flächen 1. bis 5. Ordnung verwendet. Algebraische Flächen höherer als zweiter Ordnung sind bislang nicht allgemein in Monte Carlo Codes implementiert.

Eine mit Hilfe externer Software erstellte Triangulation der Bauteilflächen, ausgeführt zwischen CAD und EIRENE, erzeugt unter Berücksichtigung der gewünschten Detailtreue der Approximation ein Dreiecksmodell, dessen Flächen 1. Ordnung für EIRENE direkt nutzbar sind.

Im zu optimierenden Modul `TIMEA.F` des EIRENE Codes werden bislang die Schnittpunkte der geraden Teilchenflugbahnen mit den Geometrieflächen durch eine lineare

[3] www.eirene.de/manual

Suche in allen Flächen ermittelt. Die grundlegende Idee zum Ersatz dieser Suche durch effektivere Algorithmen beruht in der Analogie von Schnittpunkt-Suche in der Teilchenverfolgung von EIRENE und der Schnittpunkt-Suche in der Strahlverfolgung von Raytracing in der Bildverarbeitung. Diese Analogie der Problemstellung erlaubt den Ansatz, Optimierungstechniken für Raytracing auf EIRENE zu übertragen.

Neben einer kurzen Einführung in allgemeine Grundlagen des Raytracings und Vorstellung von Optimierungstechniken (kD-Bäume, BSP-Bäume, Octrees, Boundary Volumes, Uniform Grids) werden die Analogien zwischen Laufzeit-Optimierung von Raytracing und der Schnittpunkt-Berechnung von EIRENE im Detail aufgezeigt.

Als geeignete Strategie werden Octrees identifiziert. Für die Punkt- und Nachbarsuche im Octree wird ein Top-Down-Ansatz auf Basis von Location Codes [1] gewählt. Die erstellte Fortran-Implementierung wird dokumentiert, die Abänderungen gegenüber der linearen Suche nach Schnittpunkten aufgezeigt und das Laufzeitverhalten der Simulation bei Nutzung der Erweiterung untersucht.

Eine Reduktion der Laufzeit um mindestens 50%, typisch aber noch deutlich signifikanter, in Abhängigkeit von der Komplexität der Modell-Geometrie ist gegeben, und Ansätze zur weiteren Verbesserung werden aufgezeigt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

In der Fusionsforschung werden neben experimenteller Forschung an magnetisierten Hochtemperaturplasmen auch umfangreiche Computerprogramme zu deren Interpretation, aber auch zur Extrapolation auf künftige Kraftwerke entwickelt.

Eine solche Software ist der Monte-Carlo-Transport-Code EIRENE, der am Institut für Energie- und Klimaforschung 4 des Forschungszentrum Jülich seit mehreren Jahrzehnten stetig weiter entwickelt wird (siehe www.eirene.de, dort insbesondere auch unter „relevant reports“).

Das Grundprinzip von Monte Carlo Codes ist seit vielen Jahrzehnten praktisch unverändert geblieben, allerdings sind die Anwendungen, insbesondere die in der Simulation erfassten geometrischen Details, immer komplexer geworden.

Der EIRENE-Code beschreibt den Transport von zumeist neutralen Teilchen (auch: Photonen) in nahezu beliebigen 3D Modellgeometrien, häufig allerdings zugeschnitten auf Fragestellungen in einem Fusionsexperiment oder für einen künftigen Reaktor (ITER) oder Kraftwerk (DEMO). Das Hintergrundmedium (meist Elektronen und Ionen), in dem der Transport der Teilchen studiert wird, wird typischerweise durch andere Codes mittels Flüssigkeitsnäherung simuliert. Mit dem EIRENE-Code können sowohl Vakuumbedingungen als auch Plasmabetrieb simuliert werden. Das zu simulierende Modell wird beschrieben durch die Geometrie (Ränder) , die Auswahl der zu simulierenden Teilchen, und die im Inneren der Geometrie und an den Wänden ablaufenden Prozesse. Alle Parameter für das Modell werden mittels einer formatierten Eingabedatei an EIRENE übergeben. Die in dieser Datei enthaltenen Modelle zur Beschreibung der Modellgeometrie können (unter anderem) aus Flächen 1. und 2. Ordnung im $\mathbb{R}^3$ bestehen (sog. „Additional Surfaces“) und bilden ein so genanntes „Boundary Representation“ (BREP) Modell. Ein vollständiger Satz an Flächensegmenten in einer EIRENE Modellgeometrie kann aus einer Kombination von sog. regulären Gitterflächen („standard surfaces“) und „additional surfaces“ bestehen, siehe Abbildungen 1.1 und 1.2 für ein frühes Beispiel (Mitte der achtziger Jahre, aus TEXTOR-ALT-II Pumplimiter Studien mit EIRENE).

Im aktuellen Entwicklungsstand des EIRENE Codes wird nach jeder Richtungsänderung eines Teilchens, also nach jeder Reaktion im Plasma oder Reflektion an der Berandung der Modellgeometrie, jede Fläche der Geometrie auf einen möglichen Schnittpunkt mit der neuen Teilchenflugbahn untersucht. Diese lineare Suche ist bei großen Modellgeometrien (viele Elementarflächen) zunehmend ineffektiv: Eine Simulation wird meist mit 100.000 bis mehreren Millionen Teilchenverfolgungen gerechnet, um eine statistisch signifikante Aussage über die Abläufe in der Realität treffen zu können. Es sind daher viele Schnittpunkt-Tests auszuführen, die zu einem großen Teil redundant sind, weil ein Teilchen aufgrund seiner geraden Flugbahn während einer einzelnen Verfolgung gar nicht alle Flächen der Geometrie treffen kann.

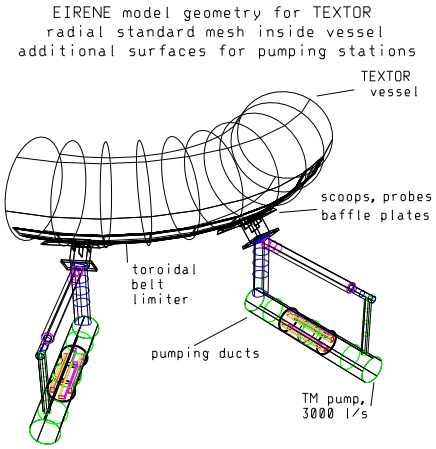


Abbildung 1.1.: Frühes Beispiel einer kombinierten BREP und VOXEL Diskretisierung des Rechengebietes in EIRENE, ca. 1985, für ALT-II Pumplimiter Studien an TEXTOR. Diskretisierung innerhalb des TEXTOR Tokamak Vacuumgefäßes (Torus): VOXEL (reguläres Gitter), ergänzt um ca. 100 (BREP) algebraische Flächen erster und zweiter Ordnung zur Beschreibung der ALT-II Scoops, Pumpducts und innerer Strukturen, z.b. Sonden, Getterpumpen, etc..

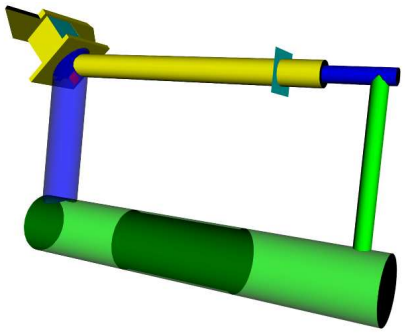


Abbildung 1.2.: wie linke Abbildung, aber nur BREP Teil der Geometrie, und unter Verwendung eines neueren 3D Visualisierungsmoduls, ca. 2009

Die momentane „Brute-Force“-Methode ist also im Falle einer massiven Erhöhung der Flächenanzahl durch höhere Detailgenauigkeit oder größere Geometrien aufgrund des dann zu stark anwachsenden Rechenaufwands problematisch. Im Kontext dieses Reports setzen sich aktuelle, aus CAD gewonnene Modellgeometrien, aus besonders vielen Dreiecken (1.000 - 20.000 Stück, oder mehr) zusammen, siehe z.B. Abb. 1.3.

Die Teilchenverfolgung in EIRENE ist ähnlich der Problemstellung der Strahlverfolgung im sogenannten Raytracing (Bildverarbeitung). Werden dort Strahlen auf ihrem Weg durch eine 3D-Szene verfolgt und auf Schnittpunkte mit den Objekten der Szene untersucht, ist die 3D-Szene in EIRENE mit der Modellgeometrie der Simulation gleich zu setzen.

Der vorliegende Report beschreibt die Entwicklung einer Erweiterung bzw. Optimierung von EIRENE, die es erlaubt, detailgetreue Bauteile von Simulationsmodellen in deutlich kürzerer Rechenzeit als bisher zu simulieren. Bei gleichem Rechenaufwand können damit mehr Monte Carlo Teilchen verfolgt werden und die Ergebnisse werden statistisch präziser.

Dazu soll zunächst die mathematische Analogie zwischen Raytracing und Teilchenverfolgung abgegrenzt werden, um zu prüfen, ob Optimierungstechniken für Raytracing auf EIRENE übertragbar sind. Nachdem dieses positiv beantwortet ist, soll die geeignetste Strategie identifiziert und implementiert werden. Lineare Suche und Erweiterung werden schließlich bzgl. ihres Laufzeit-Verhaltens untersucht und verglichen. Von der Implementierung der Erweiterung wird eine Reduktion von mindestens (im ungünstigsten Fall) 50% relativer Laufzeit unabhängig von der zu simulierenden Geometrie erwartet.

Das nächste Kapitel führt genauer in die Materie und Problemstellung ein. Anschließend erfolgt die Darstellung, wie die Kombination von Monte-Carlo-Simulationen und Optimierungstechniken für Raytracing die Simulationen beschleunigen können. Das vierte Kapitel dokumentiert die konkrete Implementierung des Verfahrens, gefolgt von Tests auf Korrektheit und den Effekt der Verbesserung im fünften Kapitel.

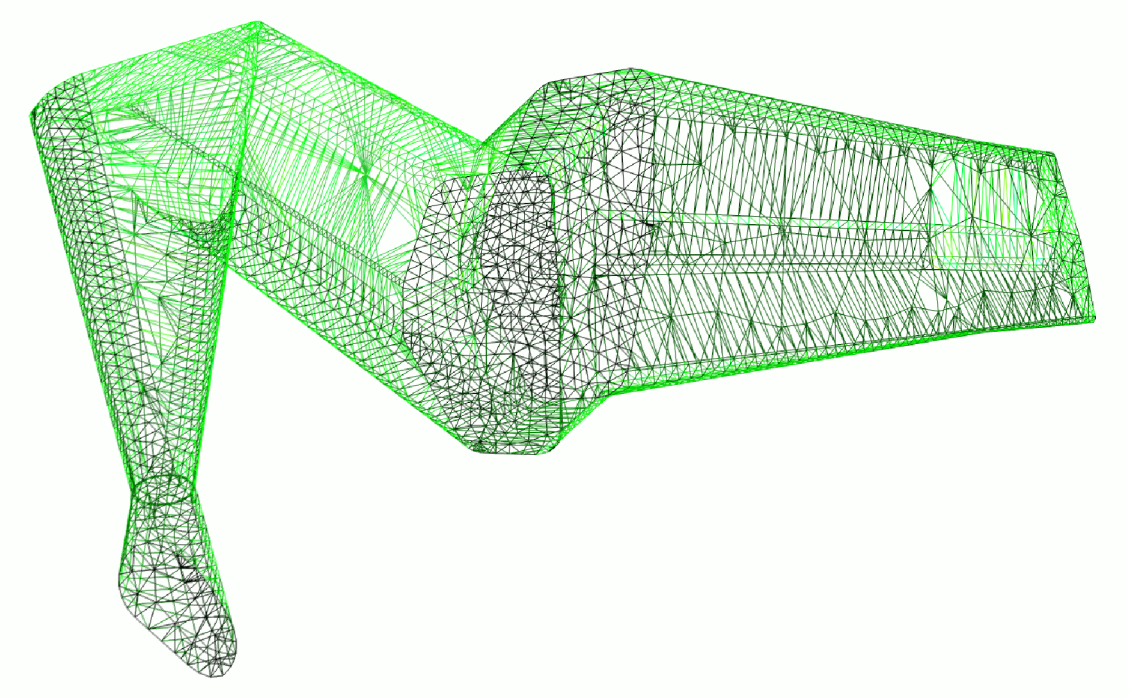


Abbildung 1.3.: Geschlossenes Dreiecks-Modell eines Diagnostik -„Port-Plugs“ des ITER Fusionsreaktors. Darstellung als Drahtgittermodell, insgesamt 7161 Dreiecke direkt aus der EIRENE BREP-Option („Additional Surfaces“) generiert.

2. Hintergrund und Analyse

In der Monte-Carlo-Simulation werden Objekte, in unserem Fall meist elektrisch neutrale Teilchen (im Folgenden kurz „Teilchen“ genannt) bei ihrem Flug durch eine vorgegebene geometrische Anordnung verfolgt. Dabei starten die Neutralteilchen an ihrem Entstehungsort mit einer zufällig gewählten Richtung und Geschwindigkeit und fliegen auf einer Geraden, bis sie auf ein „Hindernis“ treffen. Ein Hindernis kann ein anderes Teilchen oder auch eine Fläche der umgebenden Reaktionskammer sein. Geladene Teilchen werden von EIRENE nur unter stark vereinfachenden Annahmen verfolgt, da ihre Flugbahnen keine Geraden darstellen und auch die Modellierung der physikalischen Reaktionen sich deutlich von denen für neutrale Teilchen unterscheidet. Prinzipiell gilt aber alles hier Gesagte sowohl für neutrale also auch geladene Teilchen in externen Kraftfeldern. Wir beschränken uns also hier auf den Transportalgorithmus von neutralen Teilchen (gerade Flugbahnen).

Wird dann bei einer Reaktion eines neutralen Teilchens ein geladenes Teilchen frei gesetzt, wird dieses in nur der statistischen Auswertung vermerkt und die Simulation des neutralen Teilchens beendet, oder fortgesetzt, je nachdem ob nach dem Stoßereignis noch ein (oder mehrere) neutrales Teilchen.

Während des Fluges und beim Aufprall auf ein Hindernis können verschiedene Reaktionen auftreten. Aus einem einzelnen Teilchen können mehrere entstehen oder dieses kann an einer Fläche reflektiert oder absorbiert werden. Für die realistische Simulation können Materialeigenschaften der die Modellgeometrie berandenden Bauteile, das Hintergrundmedium (Plasma) und die darin stattfindenden Reaktionen (Stoßprozesse) spezifiziert werden. Das Hintergrundmedium (Plasma) wird zumeist mittels eines dreidimensionalen Gitters beschrieben, das durch verschiedene Flächen begrenzt wird. Dieses sogenannte „Standardrechengitter“ wird durch Diskretisierung des gesamten betrachteten Raumvolumens generiert und aus z.B. Tetraedern, Quadern, etc. zusammengesetzt (VOXEL-Algorithmus) (siehe [2]).

Die das Gitter begrenzenden Flächen (aber ggfls. auch innere Flächen) werden unter Anderem durch zusätzliche, im Standardgitter nicht enthaltene algebraische Flächen (sog. „Additional Surfaces“) unter Nutzung eines „Boundary Representation“ (BREP)-Modells definiert, indem diese Flächen der Geometrie über ihre mathematischen (algebraischen) Gleichungen angegeben werden. Diese Flächen können Flächen 2. Ordnung (Zylinder, Kugeln, etc.) und Flächen 1. Ordnung (Ebenen) sein. Dabei sind die Ebenen entweder durch zwei Punkte einer Gerade und eine dritte, ignorierbare, Koordinate definiert, oder aber durch drei bis fünf Punkte im Raum - die dann ein ebenes Dreieck, Viereck oder Fünfeck im Raum bilden - oder durch die explizite Angabe der Koeffizienten der Ebenengleichungen. Im Falle der Zweipunkteform ist die Ebene automatisch parallel zu einer der Koordinatenachsen, die durch den Nutzer in der Eingabe steuerbar ist.

Werden krummlinige Koordinaten verwendet, sind diese Flächen dann auch entsprechend gekrümmt.

Alle Schnittpunktberechnungen von fliegendem Teilchen und Flächen werden analytisch (sogar: exakt algebraisch) durchgeführt, um Rechenungenauigkeiten durch numerische Verfahren zu vermeiden. Aus diesem Grund ist es nicht möglich, Flächen 5. (und höherer) Ordnung zu implementieren, da keine analytische Lösung des Gleichungssystems möglich ist. Flächen 3. oder 4. Ordnung sind bislang nicht implementiert, da die Anwendung der Cardanoschen Formeln recht aufwändig wäre und dies bislang nicht benötigt wurde. Alle Flächen mit höherer als 2. Ordnung müssen mit Hilfe geeigneter Verfahren durch Flächen 1. und 2. Ordnung approximiert werden: Ein Beispiel ist die Approximation durch Triangulation (s. u.). Die Modellgeometrie ist während der Monte Carlo Simulation statisch, d.h. es gibt keine Veränderungen in Anordnung oder Eigenschaften der Flächen während eines Laufs.

Die konkrete Fragestellung für die vorliegende Arbeit leitete sich aus einem Auftrag zur EIRENE-Simulation von sogenannten Diagnostik „Port-Plugs“ für den derzeit in der Bauphase befindlichen Fusions-Reaktor ITER ab. Ein Port-Plug ist ein Anbau an einem Fenster in der Wand des Reaktors, der z. B. zur Ankopplung von Diagnostiken an das Reaktionsvolumen genutzt wird (siehe Abb. 1.3). Bei der Konstruktion wird, wie im Maschinenbau üblich, 3D-CAD-Software benutzt, anhand derer die Fertigung betreffend Form, Material und Beschaffenheit exakte Vorgaben zur Herstellung erhält.

Der Entwurf eines Bauteils für Fusionsreaktoren ist ein Prozess, an dem ein - häufig auch internationaler - Personenkreis aus Physikern, Maschinenbauern, Konstrukteuren, etc. beteiligt ist. Das Spektrum der begleitenden numerischen Untersuchungen reicht dabei von der Simulation eines echten Plasmas in einem Reaktor über die Behandlung eines Teilausschnitts bis zur Simulation von einzelnen Teilchen in einem Peripherie-Modul (wie den genannten „Port-Plugs“) unter Vakuumbedingungen.

Die Fragestellung der Vakuum-Simulation von zu konstruierenden Bauteilen von Fusionsreaktoren ist nicht neu. Zu diesem Thema wurden bereits in den achtziger Jahren viele Optionen des EIRENE Codes entwickelt, so auch die BREP Darstellung von komplexen 3D geometrischen Anordnungen (Abb. 1.1). Neuere Ansätze wie die Ankopplung von EIRENE an CAD Systeme wurden z.B. in der Diplom-Arbeit [3] verfolgt. Der darin beschriebene Ansatz beruhte darauf, mit Hilfe eines bestehenden Konverters aus CAD-Ausgabedaten zunächst Eingabedaten für den Neutronen-Transportcode „Monte Carlo Neutral Particle“ (MCNP)[1] zu erzeugen. Dieses Vorgehen wurde gewählt, da beide Codes, MCNP und auch EIRENE, das oben bereits genannte BREP Konzept zur Diskretisierung des Rechengebietes und dessen Umrandungen nutzen. nutzen. Dieser Konverter wurde dahingehend erweitert, dass die MCNP-Eingabe in ein für den EIRENE-Transportcode nutzbares Format umgewandelt wird. Alle Flächen aus dem CAD-Programm werden dabei zunächst in unbegrenzte Flächen 1., 2. oder - mit dem Torus als Spezialfall - 4. Ordnung gewandelt, was den Eingabemöglichkeiten von MCNP entspricht. In MCNP werden diese Flächen zur Laufzeit mit Hilfe von kombinatorischen Ausdrücken zu sog. „Zellen“ zusammen gesetzt, sodass zur Simulation ein geschlossenes Modell entsteht.

[1] MCNP: http://mcnp-green.lanl.gov/

Zwar benutzt auch MCNP über diesen Umweg einer kombinatorischen Geometrie ein BREP-Modell, jedoch ist die Überführung in EIRENE nicht direkt möglich, da EIRENE direkt mit begrenzten Flächen als Eingabe arbeitet. Eine Erweiterung des EIRENE-Codes in der Weiterentwicklung der Diplomarbeit in [3] sah vor, wie der MCNP-Code unbegrenzte Flächen kombinatorisch zu verknüpfen, um Zellen zu erhalten. Die konkrete Umsetzung dieses Ansatzes wurde wieder aufgegeben, da der Umweg von den begrenzten CAD Flächen über unbegrenzte kombinatorisch zusammengesetzte Flächen, um dann doch wieder die in EIRENE ohnehin verfügbaren begrenzten Flächen zu erhalten, zu sehr komplexen Algorithmen führt, nicht zuletzt aber auch auf Grund der fraglichen freien Verfügbarkeit eines Konverters von CAD nach MCNP.

Ein neuer Ansatz führt nun direkter von CAD in EIRENE. Er beruht auf dem Export von sogenannten STEP-Dateien, die der ISO-Norm 10303 folgen [4]. Dieses Datenformat wird von den meisten in der Konstruktion verwendeten namhaften CAD-Programmen (CATIA, AutoCAD Inventor, etc.) unterstützt, wodurch eine gemeinsame Basis für den Austausch von Modellen gegeben ist. Das STEP-Datenformat ist in der Norm ISO 10303 beschrieben, die in Anwendungsprotokollen (engl. „application protocols“, AP) die Datenstrukturen definieren, die zum Austausch des Modells notwendig sind.

Interessant ist hier vor allem das AP-214 (ISO 10303-214), das von 3D-CAD-Programmen wie CATIA als STEP-Exportmöglichkeit vorgesehen wird. Mit diesem AP können verschiedene Arten von 3D-Modell-Repräsentierungen genutzt werden: neben „Constructive Solid Geometry“ (CSG) und „Wireframe“ wird die „Boundary Representation“ (BREP) als Beschreibung des Modells ermöglicht. Die für den EIRENE-Code interessante Variante ist die BREP-basierte Modellierung, da diese bereits der Modellierung von Geometrien durch begrenzte mathematische Flächen innerhalb des EIRENE Codes entspricht. In den für diese Arbeit analysierten Beispieldaten wurde eben diese Möglichkeit der CAD-Software genutzt (CATIA STEP-Export).

Bei der Analyse der STEP-Daten fällt auf, dass von den CAD-Programmen B-Spline-Flächen bis fünfter Ordnung verwendet werden. Diese sind nicht direkt in EIRENE-Eingaben abbildbar, da der Code nur Flächen maximal 2. Ordnung unterstützt. Bevor die 3D-Modelle in EIRENE bearbeitet werden, müssen sie daher mittels eines Konverters in kompatible Flächen zerlegt werden. Im konkreten Fall wird ANSYS[2] als Triangulierer gewählt, sodass die B-Spline-Flächen in Dreiecksgitter zerlegt wurden.

Ein im Institut für Energie - und Klimaforschung - Plasmaphysik des Forschungszentrum Jülich entwickeltes MATLAB-Programm realisiert diese Konvertierung von STEP-Daten mit ANSYS in EIRENE-Eingaben. Bei der aktuellen Problemstellung der Simulation der Port-Plugs stellt sich der Umweg über einen Triangulierer als deutlich einfachere und besser unterstützbare Möglichkeit dar. Im Gegensatz zu den kombinatorisch mit Flächen begrenzten Zellen von MCNP (vgl. Ansatz von [3]) ist die Übersetzung des Modells in Dreiecke durch bereits verfügbare Software vereinfacht für die Nutzung in EIRENE anwendbar, ohne dass der Code Kombinatorik von Flächen beherrschen muss.

[2] ANSYS: http://www.ansys.com

Um die notwendige Detailtreue des Modells zur adäquaten Simulation zu erhalten, muss die Anzahl der Dreiecke hinreichend groß gewählt werden. Die vorliegende Arbeit ist in dem Kontext entstanden, EIRENE der Anforderung der Simulation von Geometrien mit größeren Mengen von „Additional Surfaces“ (10^3 bis 10^6 Flächen) anzupassen. Die Optimierung von EIRENE soll sich dabei darauf beschränken, die Anzahl der Schnittpunkttests zu begrenzen, die für jedes Teilchen auf seiner Flugbahn durchgeführt werden.

Der Ansatz zur Optimierung des Algorithmus stützt sich auf die Ähnlichkeit der Problemstellung zur Optimierung von Raytracing-Algorithmen aus der Gruppe der bilderzeugenden Verfahren. Das nächste Kapitel soll daher zunächst Raytracing vorstellen, zeigen wie es optimiert werden kann und welche Beziehung zur eigentlichen Problemstellung vorliegt.

3. Raytracing: Grundlagen und Optimierung

Unter „Raytracing“ versteht man allgemein ein Verfahren aus der Gruppe der bilderzeugenden Algorithmen [5, 6]. Grundlage des Verfahrens ist die Strahlverfolgung (daher auch der englische Name). Ziel des Algorithmus ist, eine dreidimensionale Szene auf eine zweidimensionale Projektionsebene abzubilden. Die Lage, der Abstand und der Winkel der Projektionsebene im Verhältnis zur 3D-Szene bestimmen den später sichtbaren Bildausschnitt. Die Ebene wird auf einen rechteckigen Ausschnitt mit einem vorher festgelegten Verhältnis der Seitenlängen begrenzt und mit Hilfe eines Gitters unterteilt, dessen Gitterabstände durch Größe des Ausschnitts und geforderte Auflösung des Bildes vorgegeben werden.

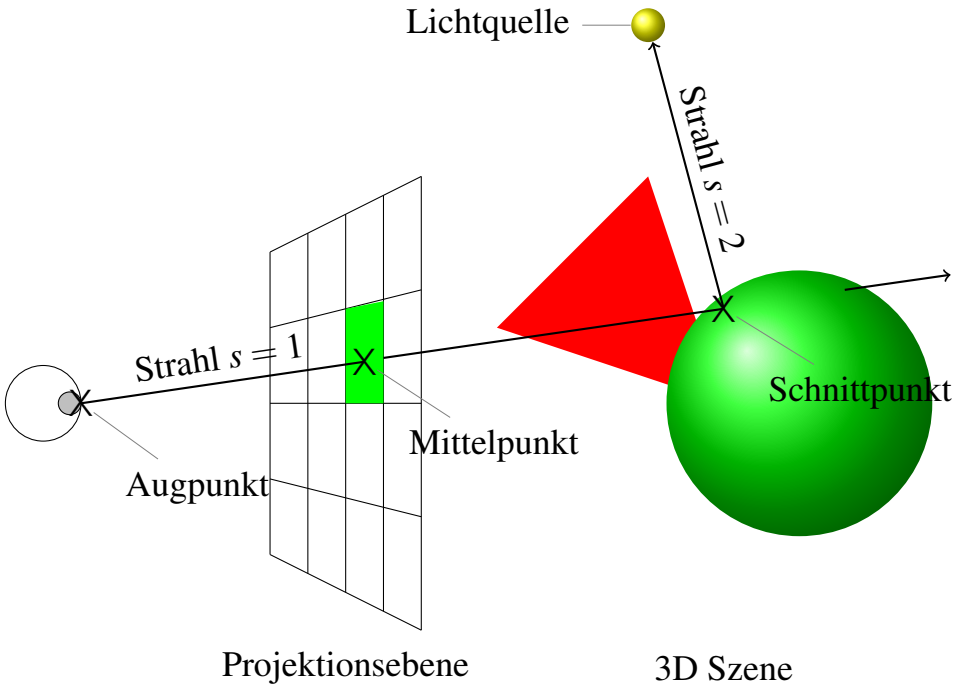


Abbildung 3.1.: Prinzip des Raytracing. Eine Gerade (Strahl, Rekursionsstufe $s = 1$) durch Aug-Punkt und Mittelpunkt eines Rechtecks in der Ebene wird mit den Objekten der Szene geschnitten. Weitere Rekursionsstufe ($s = 2$) mit Strahl von Schnittpunkt zur Lichtquelle.

Zur Projektion des Bildes werden von einem Punkt vor der Ebene („Aug-Punkt“) Strahlen als Gerade durch jeden Bildpunkt der Projektionsebene (Mittelpunkt eines Rechtecks im

Gitter) in die Szene gelegt (siehe Abbildung 3.1). Diese treffen auf die Szene hinter der Ebene.

Zu jeder Geraden wird ein Schnittpunkttest mit jedem Objekt der Szene ausgeführt. Der Schnittpunkt, der am nächsten zur Ebene liegt, gehört zu dem Objekt, das am nächsten zu ihr liegt. Die Farbe des späteren Bildpunktes wird daher primär von der Farbe dieses Objektes bestimmt.

Sekundär bestimmen Belichtungsmodelle Abweichungen von der Objektfarbe. Mit verschiedenen, kombinierbaren Belichtungsmodellen ist Raytracing dazu geeignet, Pseudo-3D-Bilder zu berechnen, die vom Betrachter als sehr realtitätsnah empfunden werden können. Die Belichtungsmodelle simulieren die realen Effekte von Licht und Schatten, Lichtreflexion, Lichtbrechung, Lichtbeugung etc.

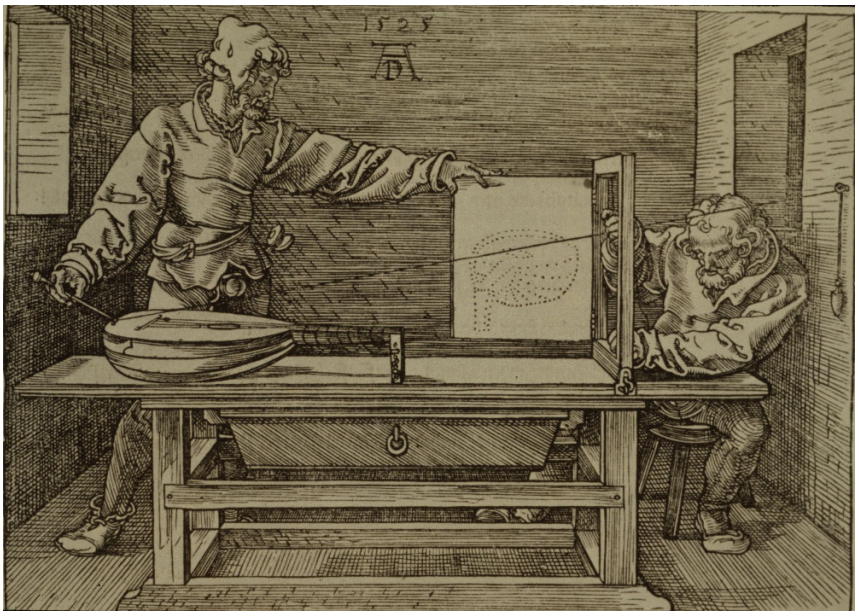


Abbildung 3.2.: Albrecht Dürer, „Der Zeichner der Laute“, 1525, Quelle: http://www.usc.edu/schools/annenberg/asc/projects/comm544/library/images/626.html

Das Raytracing-Verfahren ist in ähnlicher Weise bereits auf einem Bild von Albrecht Dürer von 1525 zu erkennen (siehe Abbildung 3.2). Es wurde in der Informatik das erste Mal von Appel im Jahr 1968 beschrieben (siehe [7]), jedoch damals als eine eher theoretische Möglichkeit aufgrund der sehr begrenzt möglichen Rechenkapazitäten [5].

Insbesondere dann, wenn nicht nur primär die Farbe des Punktes berechnet werden soll, sondern ebenfalls sekundäre Effekte (Schatten, Lichter, Reflektionen, Lichtbeugung, etc.), ist der Raytracing-Algorithmus im Schnittpunkt rekursiv anzuwenden (siehe [8]).

Dazu werden Strahlen von jedem Schnittpunkt zu den in der Szene definierten Lichtquellen verfolgt. Es sind erneut Schnittpunkte mit Objekten zu ermitteln, um die durch Verdeckung vorhandenen Abschattungen berechnen zu können. Sind Lichtreflektionen

von Objekten gewünscht, müssen von diesen Schnittpunkten wiederum Strahlen zu den Lichtquellen geschossen werden, und so weiter.

Diese rekursiven Aufrufe des einfach zu implementierenden Algorithmus werden „teuer“ im Sinne eines hohen Maßes an notwendiger Rechenkapazität: Für jeden Rekursionsschritt sind Tests mit n Objekten der Szene zu berechnen, was dem Aufwand $O(n)$ entspricht. Es sind m Geraden (m entspricht der Anzahl der Bildpunkte) von Aug-Punkt und Bildpunkt zu testen. Für eine Tiefe s der Rekursion ergibt sich damit ein Aufwand der Klasse $O(m * n^s)$ für die Schnittpunktberechnungen mit analytischer Lösung der Gleichungen bis maximal Flächen vierter Ordnung. Um Raytracing zu optimieren, liegt es daher nahe, die Anzahl der Schnittpunkttests zu reduzieren, indem entweder das Bild kleiner wird (meist unerwünscht), weniger Rekursionsstufen s genutzt werden (unrealistischeres Bild) oder die Anzahl der zu testenden Objekte n pro Stufe verringert wird.

3.1. Analogie von Raytracing und Teilchenverfolgung in EIRENE

Die Problemstellung bei dem Monte-Carlo-Teilchen-Transport-Code EIRENE ist sehr ähnlich. Auch hier wird ein Teilchen von einem Punkt aus in die „Szene“ (die Geometrie) geschossen. Statt m Zweipunkt-Geraden aus einer Projektionsebene mit m Bildpunkten und einem Aug-Punkt werden in EIRENE m Teilchen mit Startpunkt und Richtungsvektor als Flugrichtung des Teilchens mit daraus resultierenden m Geraden verfolgt. EIRENE hat immer eine vom Benutzer vorgegebene obere Schranke der zu verfolgenden Teilchen, sodass diese Schranke mit m gleichgesetzt werden kann.

Die Rekursionsstufe s drückt in EIRENE die Anzahl der Ablenkungen oder Reflektionen eines Teilchens aus, bevor es aufgrund simulierter Prozesse nicht weiter verfolgt wird: An den Geometrieflächen oder im Plasma können durch den Nutzer vorgegebene physikalische Prozesse ablaufen, sodass aus einem Teilchen mehrere werden, die verfolgt werden müssen, oder Teilchen durch eine Reaktion/Absorption „verschwinden“. Welcher Fall davon eintritt, bestimmt ein Monte-Carlo-Experiment. Da es im Gegensatz zu normalem Raytracing für 3D-Szenen keine Lichtquellen gibt, zu denen Strahlen gesendet werden können, wird in jeder Rekursionsstufe der Richtungsvektor einer neuen Teilchenflugbahn anhand des Experiments ausgewürfelt. Ortsvektor der Geraden der Flugbahn ist dabei z. B. der Schnittpunkt von vorheriger Teilchenflugbahn und einer Fläche.

Entscheidend ist daher auch hier, die Anzahl der Schnittpunkttests mit den n Flächen der Geometrie zu senken, da die Anzahl m vorgegeben ist und eine statische Beschränkung der Rekursionstiefe s zur allgemeinen Reduktion der Laufzeit nicht sinnvoll ist. Im Folgenden sollen nun Optimierungstechniken vorgestellt werden, die im Raytracing genutzt werden. Ferner sollen sie auf ihre Eignung für EIRENE untersucht werden.

3.2. Laufzeit-Optimierung von Raytracing

Für die Beschleunigung der Verfolgung ist es von Bedeutung, Flächen und komplexere Objekte aus diesen Flächen so zu organisieren, dass bei der Verfolgung des Teilchens möglichst wenige Flächen oder Objekte getestet werden müssen. Ist es im Vorfeld bei der Erstellung der Szene nicht möglich, Objekte zu vereinfachen - z. B. einen Zylinder nicht mit Dreiecken, sondern als Fläche 2. Ordnung und zwei Ebenen darzustellen - so gilt es, die im Raum verteilten Flächen möglichst geschickt anhand einer zu findenden Strategie einzuteilen. Die globale Suche nach Schnittpunkten kann durch ein Divide-and-Conquer-Verfahren in das kleinere Teilproblem einer lokalen Schnittpunktsuche zerlegt werden (Divide-Schritt). Der Schnittpunkt mit der geringsten Entfernung zum Startpunkt ist später aus der Menge der Kandidaten aus den lokalen Suchen effektiv herauszusuchen (Conquer-Schritt), sodass das Gesamtproblem durch günstiger lösbare Teilprobleme gelöst werden kann.

Der Divide-Schritt benötigt eine adäquate Datenstruktur, auf der das Teilproblem gelöst werden kann. Diese Datenstruktur kann auf verschiedene Arten gestaltet werden, je nach Anforderung an den Algorithmus.

Die Arbeit in [5] erläutert die einzelnen Strukturen und die Verfahren zu ihrer Anwendung als kompakte Übersicht. Die dort vorgestellten Strukturen und Ansätze lassen sich verschiedenen Schwerpunkten zuordnen: flache und hierarchische, raumorientierte und objektorientierte Strategien (vgl. Abb. 3.3). Die einzelnen Datenstrukturen, die in der Abbildung 3.3 genannt werden, werden im weiteren auf ihre Eignung für die Anwendung in EIRENE untersucht.

Keine dort angegebene Datenstruktur ist fest an eine Ausprägung eines Ansatzes gebunden. Mischformen verschiedener Datenstrukturen und Kombination von Ansätzen zu bilden ist möglich und nach Anwendungsfall verschieden. Gegebenenfalls ist eine solche Mischform für die Anwendung in EIRENE zu verwenden.

Allen raumteilenden Datenstrukturen (und eventuellen Hybriden) ist gemein, dass der Strahl nach dem Eintritt in den Bereich, über dem die Datenstruktur definiert ist, durch diesen verfolgt werden muss. Dies nennt man „Ray Traversal“ (engl.: Strahlverschiebung).

Der Begriff entstammt dem Ablauf des Divide-Schrittes: Es wird lokal mit dem Strahl nach Schnittpunkten gesucht. Wenn keine Punkte zu finden sind - aus welchem Grund auch immer - wird der Strahl in das in Strahlrichtung benachbarte Teilgebiet verschoben um dort lokal zu suchen.

Die Schwierigkeit zur Auffindung des benachbarten Teilgebietes ist bereits beim Entwurf der Datenstrukturen zu betrachten, denn die Suche nach diesem Nachbarn muss möglichst unaufwändig sein, um den Kostenvorteil durch die verringerte Anzahl von Tests nicht durch die Nachbarsuche aufzuzehren.

3.2.1. Flache Strukturen

Flache Datenstrukturen haben den Vorteil, dass sie sehr einfach zu verwalten sind. Es ist nicht notwendig, Zeiger auf untergeordnete Teilbereiche zu verwalten, denn es gibt keine.

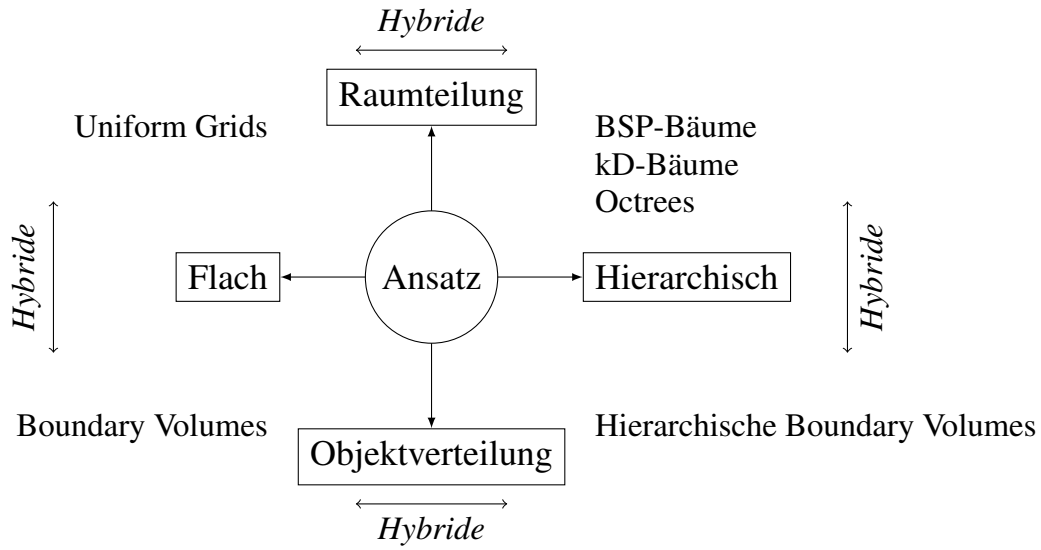


Abbildung 3.3.: Übersicht der Daten-Strukturen zur Beschleunigung von Raytracing nach [5].
Die Strukturen sind dort eingezeichnet, wie sie hauptsächlich genutzt werden. Hybride Mischformen und Abwandlungen sind möglich.

Die objektorientierte Strategie der „Boundary Volumes“ (engl., ungefähr: eingrenzende Volumen, im weiteren: BVs) versucht primär mit einer flachen Datenstruktur die Schnittpunkt-Tests zu vereinfachen (und ggf. dadurch zu reduzieren). Nur komplex testbare Objekte der Szene (ggf. aus mehreren Flächen bestehend) werden durch einfacher testbare Körper (Quader, Kugeln oder Polyeder) abstrahiert, wobei diese Körper das Objekt vollständig umhüllen. Es ist auch möglich, mit Hilfe von Boolschen Operationen verschiedene Hüllkörper zu schneiden, um die Abstraktion des Objektes zu verbessern und unnötigen, zusätzlichen Leerraum des Hüllkörpers zu vermeiden. Wenn der Strahl das genäherte Körpervolumen trifft, wird der komplexe Test am eigentlichen Objekt vollzogen, um den exakten Schnittpunkt zu bestimmen. Trifft der Strahl nicht, ist an Rechenzeit für das komplexe Objekt nur der Test der Hülle aufgewendet worden. Diese Strategie lässt sich gut anwenden, wenn komplexe Objekte mit geringem Rechenaufwand abstrahiert werden können. Es spielt dabei zunächst keine Rolle, wie die Objekte im Raum verteilt sind. Eine Ausnutzung von räumlichen Anordnungen ist nicht möglich.

Im Gegensatz dazu sind „Uniform Grids“ (engl., ungefähr: gleichmäßige Gitter) stark räumlich orientiert. Der Raum wird dabei anhand einer vorher festgelegten Größe (N_i) mit achsenparallelen Gitterlinien in regelmäßige Abschnitte unterteilt. Die Gitterzellen erhalten Indizes, welche ihren Abstand vom Ursprung in den XY-, XZ- und YZ-Hauptebenen ausdrücken. Die Objekte im Raum werden anhand ihrer Lage mit den entsprechenden Gitterzellen assoziiert, sodass bei einem Durchlaufen der räumlichen Struktur nur die Objekte getestet werden müssen, die in der jeweiligen Gitterzelle liegen.

Die bei Erstellung des Gitters vorzugebende Größe entscheidet dabei, wie weit ein Strahl in einem Schritt verschoben wird und ebenfalls, wie viele Objekte in eine Gitterzelle fallen. Es gilt, ein gutes Mittelmaß zwischen Gittergröße und Objektzahl zu finden, um den Strahl einerseits möglichst schnell durch den Raum bewegen zu können, andererseits die Anzahl der Schnittpunkttests mit Objekten in den Zellen so gering wie möglich zu halten.

Laut [5] kann dazu keine allgemeine Vorschrift angegeben werden, da die Größe für jeden Anwendungsfall gesondert betrachtet werden muss. Neuere Forschungen wie in [9, 10] versuchen, Optimierungen der Strategie und allgemeinere Algorithmen zur Bestimmung der Gittergröße zu finden. Statische Szenen werden jedoch effizienter unter Verwendung von hierarchischen Strukturen berechnet. Insbesondere bei ungleichmäßig verteilten Objekten verlieren „Uniform Grids“ aufgrund der zu geringen Adaptivität viele Vorteile im Vergleich zu hierarchischen Strukturen [10, 5]. Diese Strategie ist somit nicht für EIRENE geeignet, da die Szene statisch ist und nicht von einer gleichmäßig verteilten Detaillierung der Geometrie ausgegangen werden kann. Einfache Boundary Volumes sind ebenfalls ungeeignet zur Nutzung in EIRENE, da die Objekte nicht geeignet abstrahiert werden können. Die große Menge Dreiecke durch eine große Menge Kugeln zu ersetzen ist nicht effektiv, eine Zusammenfassung von Objektteilen aber auch nur schwer möglich, da dies bereits durch den Konverter geschehen müsste.

Die beiden vorgestellten Strategien können auch kombiniert werden und hybride Strukturen erzeugen. Dabei können beispielsweise Boundary Volumes (BVs) von verschiedenen Objekten mit eigenen Uniform-Grids unterteilt werden. Ebenso können in einem Uniform Grid nicht (oder nicht nur) die Ursprungsobjekte assoziiert werden, sondern deren BVs.

3.2.2. Hierarchische Strukturen

Im Gegensatz zu flachen Strukturen basieren hierarchische auf der Unterteilung der Objekte oder des Raum in kleinere Einheiten, die in größeren Einheiten gruppiert werden. Die Tiefe (Anzahl der Ebenen) und Auflösung (Anzahl der Einheiten pro Subebene) der Gruppierungen sind individuell anpassbar, sodass die Fähigkeit zur adaptiven Anpassung an nicht gleichmäßig verteilte Objekte gegeben ist. Die Parameter sind in jeder Ebene und Untereinheit individuell anpassbar, jedoch wird dadurch die Verwaltung und Nutzung komplexer.

Die objektorientierte Strategie der „Hierarchical Boundary Volumes“ (engl., ungefähr: hierarchisch eingrenzende Volumen, im weiteren: hBVs) verfolgt dabei die mehrfache Abstraktion komplexer Objekte durch Gruppierung von BVs [5, 6]. Gruppierungen auf Teilobjekten mit eigenen BVs und Gruppierungen von BVs von lokal benachbarten Objekten führen zur Bildung der Hierarchie. Schnittpunkttests erfolgen zunächst auf den BVs, in denen Objekte gruppiert sind: dies trägt zur Reduktion der notwendigen Tests durch Verringerung der Anzahl der zu testenden Objekte bei. Durch die mögliche Unterteilung der komplexen Objekte ist es möglich, weitere Reduktionen zu erzielen, da nur Teilgebiete des komplexen Objektes getestet werden müssen.

Die Implementierung der hBVs benötigt zu jedem Boundary Volume neben den Objekten, die darin eingeschlossen sein können, eine Liste der untergeordneten BVs. Daraus resultiert eine baumartige Struktur, die jedoch davon abhängt, wie Objekte als benachbart definiert werden und wieviele Nachbarn in einer Gruppe eingeschlossen werden sollen.

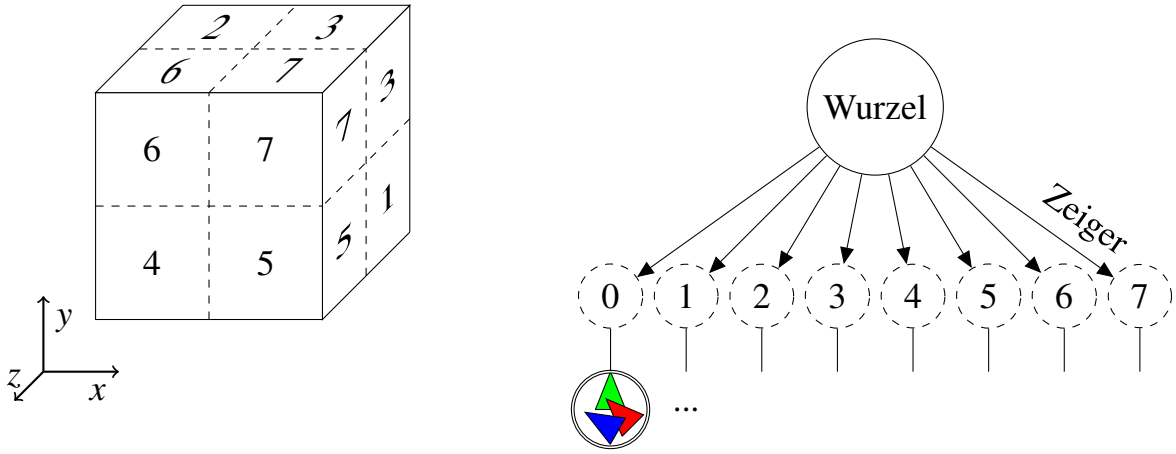
Für EIRENE sind daher hierarchische, raumorientierte Datenstrukturen besser geeignet. Bei der Gruppierung der relativ kleinen Objekte entstände eine Baumstruktur ähnlich denen der raumorientierten Strategien, jedoch ist das Kriterium zur Gruppierung unklar, da die Geometrieflächen unabhängig voneinander definiert werden. Eine Gruppierung anhand der Lage im Raum ist mit raumteilenden Strukturen sinnvoller zu lösen.

Raumorientierte Datenstrukturen, die hierarchisch organisiert sind, gibt es in verschiedenen Ausprägungen. Zwei häufig verwendete Ansätze sind „Binary Space Partition"-Bäume (BSP-Bäume) mit den „kD-Bäumen" als Sonderform, sowie Octrees.

Binary Space Partitioning (engl., binäre Raumaufteilung) zerlegt einen Raum anhand von Teilungsebenen in einen Binärbaum [11]. Eine Teilungsebene besitzt durch den Normalenvektor eine Vorder- und Rückseite, sodass die beiden Halbräume, die durch die Teilung des Raumes anhand der Ebene entstehen, einem rechten und linken Teilbaum entsprechen. Die im Raum vorhandenen Flächen werden den Unterräumen zugeordnet, in denen sie liegen. Eine rekursive Unterteilung der Unterräume bis zur Grenze, dass nur noch eine Fläche pro Halbraum vorhanden ist, erstellt den Gesamtbaum. Flächen, die nicht vollständig in einem Halbraum liegen, sondern durch die Teilungsebene geschnitten werden, werden zu zwei Flächen zerlegt, die jeweils ihrem entsprechenden Unterraum zugeordnet werden.

Die Lage der Teilungsebene ist prinzipiell frei wählbar, meist wird jedoch der Einfachheit halber eine Fläche der Geometrie als Teilungsebene genutzt [12]. Die Wahl der Fläche beeinflusst stark den Aufbau des Baumes, da die Entscheidung für eine Teilungsfläche in den Halbräumen zu mehr Zerschneidungen von Flächen führen kann als die für einer anderen Fläche. Die Gesamtzahl der Flächen im Baum ist größer als die Zahl der Geometrieflächen der Eingabedaten. Das Bestimmen der optimalen Teilungsebene, mit der die minimale Anzahl der Flächen zerschnitten werden müssen, ist somit von immenser Wichtigkeit: Steigt die Zahl der Flächen zu stark, ist die Ersparnis an Rechenzeit für die Raumaufteilung durch die erhöhte Zahl der Schnittpunkt-Tests ggf. sogar negativ, es wird somit mehr Zeit benötigt als ohne BSP-Baum.

Die Teilungen orientieren sich an den Flächen, dadurch sind Schnitte in der Regel nicht achsenparallel. Legt man fest, dass die Teilungsebenen achsenparallel verlaufen, spricht man nicht mehr von BSP-Bäumen, sondern von kD-Bäumen. kD-Bäume bilden somit eine Untermenge der BSP-Bäume. kD-Bäume sind in den Anwendungen für Raytracing weit verbreitet [13], besitzen jedoch ebenso wie BSP-Bäume die Problematik, dass sie zwar mit Hilfe von Heuristiken effizient durchlaufen werden können, jedoch dafür „gut" gebaut werden müssen. Aufgrund der vielen Wahlmöglichkeiten der Hyperebenen sind beide Baumarten anfällig für einen unbalancierten Aufbau.



(a) (b)

Abbildung 3.4.: Darstellung eines Octrees als unterteilende Octets (a) und die Entsprechung als Baumstruktur (b).
In (a) vollständig verdecktes Octet hat Index 0.
„Wurzel“ in (b) = Box in (a), Knoten in (b) = Octets in (a). Objekte jedes Octets aus (b) in (a) nicht eingezeichnet.

Erst neuere Forschungen haben effiziente Methoden gefunden, den kD-Baum mit Aufwand $O(n * log(n))$ bauen zu können [13]. Die sogenannte „Surface Area Heuristic“ zum Durchlaufen des Baumes hat sich im Bereich der Raytracings als Standardwerkzeug durchgesetzt [13, 14].

Sie setzt jedoch voraus, dass die Strahlen, die in die Szene geschossen werden, gleichförmig verteilt sind [13]. Dies ist für EIRENE so nicht anwendbar, da es keinen Aug-Punkt und keine Projektionsebene gibt, durch die die Strahlen in die Szene geschossen werden.

Aufgrund der Komplexität der Auffindung der optimalen Schnittebene in den Geometrien für EIRENE, dem damit verbundenen hohen Aufwand bei der Erstellung des Baumes und der fraglichen Anwendbarkeit der gebräuchlichen Heuristiken zum Ray Traversal, scheiden kD- und BSP-Bäume zur Verwendung aus. Eine Alternative zur Klasse der BSP-Bäume sind die sogenannten „Octrees“.

3.2.3. Octrees

Octrees sind Datenstrukturen, die ebenfalls die Strategie der hierarchischen Raumaufteilung verfolgen. Sie zerlegen im Unterschied zu den BSP- bzw. kD-Bäumen den Raum nicht anhand von Halbräumen, sondern indem ein quaderförmiges Raumvolumen in acht Unterräume aufgeteilt wird, die „Octets“ genannt werden [15]. Abbildung 3.4(a) zeigt eine solche Unterteilung. Der Mittelpunkt des Raumes wird dabei als der Schnittpunkt der drei Teilungsebenen verwendet.

Die acht Unterräume werden als Kindknoten zu einem Knoten assoziiert. Ein Knoten ist entweder Blatt und hat keine Kinder oder besitzt genau acht Kinder. Die Abbildung 3.4(b) stellt dies passend zu der Aufteilung aus Abbildung 3.4(a) dar.

Jedes Blatt beinhaltet eine Liste der Flächen, die mit ihm assoziert sind (Knoten 0 in Abb. 3.4(b) besitzt exemplarisch eine solche). Eine Fläche wird als zu einem Octet gehörig gewertet, wenn sie ganz oder teilweise in diesem liegt. Die Flächen werden nicht aufgeteilt, sondern bleiben als ganzes erhalten, sodass die Gesamtzahl der Flächen konstant bleibt. Die Menge der mit einem Octet assoziierten Flächen ist aus diesem Grund nicht disjunkt zu den Mengen anderer Octete. Der in der Implementierung genutzte Algorithmus zur Assoziierung von Flächen wird in Abschnitt 4.2.4 erläutert.

Der auf die Szene geschossene Strahl wird immer elementweise mit der Menge der Flächen eines Octets getestet. Bei einem Octet ohne auffindbaren Schnittpunkt wird anhand der durch den Strahl vorgegebenen Richtung der angrenzende Nachbarknoten gesucht und dort mit der lokalen Suche fortgefahren (Ray Traversal, s. o.).

Es sollten daher in den Octets nur wenige Flächen (Anzahl m) vorhanden sein, um die lineare Suche in den Flächen nach dem vordersten Schnittpunkt, die einen Aufwand $O(m)$ für die Suche selbst hat, nicht zu aufwändig werden zu lassen. Bei der linearen Suche sind m Schnittpunkttests mit Aufwand c auszuführen, sodass sich für die Suche nach dem Schnittpunkt pro Octet ein Aufwand $O(c*m)$ mit der entsprechenden Aufwandsklasse $O(m)$ ergibt.

Für den Aufbau des Octrees ergeben sich damit Randparameter, deren Beschränkung oder Auslegung ein gewisses Maß an „Fingerspitzengefühl“ erfordert, da es keine allgemeingültige Vorschrift zu ihrer Bestimmung gibt im Hinblick auf

1. die Höhe h des Baumes,
2. die obere Schranke a der Anzahl Flächen pro Blattknoten und
3. die maximal erlaubte Kantenlänge l[1] eines Blattknotens.

Die Parameter weisen gegenseitige Wechselwirkungen auf. Die Höhe des Baumes hängt mit davon ab, ob die maximale Kantenlänge beschränkt und die obere Schranke a klein gewählt wird. Es ist nicht zielführend, für alle Parameter feste Vorgaben zu machen, da für unterschiedliche Geometrien sehr verschiedene Konstellationen dieser Parameter sinnvoll sein können.

Es wird eine obere Schranke a definiert, da möglichst wenige Flächen in einem Blattknoten vorhanden sein sollen, unabhängig von l. Um einer starken Imbalance des Baumes entgegen zu wirken, sollte die Höhe h des Baumes ebenso beschränkt werden.

Eine starke Imbalance des Octrees hätte zur Folge, dass für eine Nachbarsuche mit Top-Down-Ansatz sehr viele Rekursionsstufen notwendig sind, da immer vom Wurzelknoten aus gesucht wird. Die von einem Blattknoten ausgehende Suche mittels eines Bottom-Up-Ansatzes kann nur bedingt Abhilfe schaffen, da bei einer zu starken Imbalance zu viele Ebenen in Richtung Wurzelknoten besucht werden müssten, sodass die Bottom-Up-Suche im schlimmsten Fall der Top-Down-Suche entspricht.

[1] l bezieht sich auf die Längen der drei Kanten des quaderförmigen Octetvolumens.

Ein Octree kann nicht nachträglich balanciert werden wie ein binärer Suchbaum, da dies eine Umordnung der Geometrie bedeuten würde. Eine alternative, objektorientierte Möglichkeit, einen Octree balanciert aufzubauen ohne die Höhe h zu beschränken, besteht aus der Nutzung eines „Mittelpunktes der Objekte“ statt des Mittelpunktes des zu teilenden Octets. Gesucht wird ein Punkt im Octet-Volumen, der als Eckpunkt die acht Unterräume so definiert, dass in jedem dieser Räume eine möglichst gleiche Anzahl an Flächen liegt.

Die Ermittlung dieses Punktes ist jedoch - ähnlich der Suche nach der optimalen Trennfläche bei BSP-Bäumen - aufwändiger als eine Auftrennung am Raummittelpunkt, wie [5] feststellt. Zudem behaupten [16], dass der optimale Punkt zwischen Raummittelpunkt und Objektmittelpunkt liegt. Weiterhin ist mit Hinblick auf eine effiziente Nachbarsuche zu bedenken, dass der in Abschnitt 4.2.3 einzuführende „Location Code“ bei einer Teilung am „Objektmittelpunkt“ nicht anwendbar ist, da bei diesem von einer Teilung in der Raummitte eines Octets ausgegangen wird.

Die Raumunterteilung mit Hilfe eines Octrees bietet durch Erhalt der Flächen auch die Möglichkeit, potentiell Flächen höherer Ordnung in ihm zu assoziieren. Flächen 2. Ordnung, die der EIRENE-Code beherrscht, in einem BSP- oder kD-Baum teilen zu müssen, ist aufwändig. Die zur Teilung genutzte Ebene müsste kombinatorisch mit der Fläche 2. Ordnung verknüpft werden, um diese zu begrenzen. Da die Implementierung von kombinatorischen Flächen vermieden werden soll, ist der Octree als weniger aufwändige erweiterbare Alternative vorzuziehen.

In Kapitel 4 ist die Realisation einer effizienten Implementierung eines Octrees für die Additional Surfaces von EIRENE zu beschreiben. Anschliessend ist diese Implementierung in Kapitel 5 neben den Laufzeitunterschieden darauf zu untersuchen, ob bei der Verwendung verschiedener, auch realer Testgeometrien unbalancierte Bäume erzeugt werden.

3.3. Baryzentrische Koordinaten

Ist zu prüfen, ob eine Gerade im $\mathbb{R}^3$ ein durch drei Eckpunkte P_1, P_2 und P_3 gegebenes Dreieck schneidet, werden baryzentrische Koordinaten genutzt. Diese Prüfung wird angewendet für die Assoziation von Dreiecksflächen mit Octeten in Abschnitt 4.2.4: durchstößt die Kante eines Octets die Dreiecksfläche, wird ein Schnittpunkt S mittels der baryzentrischen Koordinaten gefunden.

Ein Punkt P liege innerhalb eines durch die Eckpunkte P_1, P_2 und P_3 gegebenen Dreiecks. P ist dann allgemein durch die Linearkombination $P = \alpha P_1 + \beta(P_2 - P_1) + \gamma(P_3 - P_1)$ beschreibbar. Es muss für die sogenannten „baryzentrischen Koordinaten“ - siehe [17] - α, β, γ gelten, dass $0 < \alpha, \beta, \gamma < 1$ und $\alpha + \beta + \gamma = 1$ ist, da P sonst außerhalb des Dreiecks liegt.

Zur Prüfung dieser Eigenschaft für einen Schnittpunkt S einer Geraden g mit einer Ebene E, in der das Dreieck $\triangle P_1P_2P_3$ liegt, ist es hinreichend, die Parameter β und γ zu bestimmen, da S immer in der Ebene liegt, die durch die Vektoren $(P_2 - P_1)$ und $(P_3 - P_1)$ aufgespannt wird.

Ist die Gerade g mit $x = O + \lambda d$ gegeben, wird S durch Bestimmung des Parameters λ mittels eines linearen Gleichungssystems (LGS) berechnet. Die Ebene E ist gegeben mit $E : x = P_1 + \beta(P_2 - P_1) + \gamma(P_3 - P_1)$. Gleichsetzen von g und E ergibt folgendes LGS:

$$\begin{pmatrix} P_3x - P_1x & P_2x - P_1x & dx \\ P_3y - P_1y & P_2y - P_1y & dy \\ P_3z - P_1z & P_2z - P_1z & dz \end{pmatrix} \begin{pmatrix} \beta \\ \gamma \\ \lambda \end{pmatrix} = \begin{pmatrix} Ox - P_1x \\ Oy - P_1y \\ Oz - P_1z \end{pmatrix}$$

Ist $\beta + \gamma < 1$, $0 < \beta, \gamma < 1$ und $\lambda > 0$, existiert ein Schnittpunkt S innerhalb des Dreiecks $\triangle P_1P_2P_3$ in Richtung von d. Einsetzen von λ in g berechnet S.

Zur Lösung des Gleichungssystems ist es laut [18, 19] nicht sinnvoll, numerische Verfahren anzuwenden, da eine Lösung des mit drei Gleichungen sehr kleinen Systems mit Hilfe der „Cramerschen Regel“ exakt algebraisch schneller möglich ist.

4. Implementierung eines Octrees

Im vorherigen Kapitel sind die Ähnlichkeiten zwischen der Teilchenverfolgung in EIRENE und der Strahlverfolgung beim Raytracing beschrieben worden. Eine auf den EIRENE-Code angepasste Implementierung dieser Strategien ermöglicht die Beschleunigung von EIRENE. Die zu wählende Strategie muss sich adaptiv an den Raum bzw. die Objekte anpassen. Dies ist hierarchischen Strategien inherent. Zur Realisation in EIRENE wird die Strategie der raumorientierten Raumaufteilung von Octrees gewählt (vgl. Abschnitt 3.2.3).

Die Option, den Mittelpunkt zur Teilung eines Octets nicht mit dem Raummittelpunkt gleichzusetzen, sondern den Mittelpunkt durch den Objektmittelpunkt bestimmen zu lassen (siehe vorheriges Kapitel als „objektorientierte Abwandlung“), ist in dieser ersten Implementierung bewusst ausser Acht gelassen worden. Die Begründung zu diesem Schritt wird in Abschnitt 4.3.1 nachgereicht.

Voranstellend wird zunächst eine kurze Einführung in den IST-Zustand des Simulationscodes für Additional Surfaces gegeben, um darzustellen, an welchen Stellen der zu entwickelnde Code ansetzen muss. Im Anschluss daran folgt die Beschreibung der Erweiterung.

4.1. Ist-Zustand des Codes

Der EIRENE-Code ist aus verschiedenen, ineinander greifenden Modulen aufgebaut. Dazu zählen virtuelle Teilchenquellen, Geometrieberechnung, Physikmodule für Reaktionen im Plasma etc. Der für die Aufgabenstellung relevante Teil ist der sogenannte „TIMEA“-Code, als Begriff zusammengesetzt aus „TIME“ für die Flugzeit und „A“ für „Additional Surfaces“. Für die anderen in EIRENE vorhandenen Flächen gibt es ebensolche Code-Teile. Gemeinsam ist diesen Teilen des EIRENE-Codes, dass sie für ein zu verfolgendes Teilchen mit einem Startpunkt, einer Richtung und einer Geschwindigkeit als Eingaben berechnen, auf welche am nächsten gelegene Geometriefläche das Teilchen trifft. Ebenso wird der Schnittpunkt und die Flugzeit bis zu diesem berechnet. Die Eigenschaften der getroffenen Fläche entscheiden in EIRENE nicht wie im Raytracing über die Farbe eines Bildpunktes (siehe Kapitel 3), sondern bestimmen über die weitere Verfahrensweise mit dem Teilchen. Es kann absorbiert, reflektiert oder auch durch eine Reaktion in mehrere zu verfolgende Teilchen aufgeteilt werden.

Abbildung 4.1 stellt exemplarisch für die Modellierung mit „Additional Surfaces“ den Ablauf der Simulation dar, wie er zu Beginn dieses Projektes vorgefunden wurde. Dem Einlesen der Eingabedaten folgt die Initialisierung der Geometrie. Dieser Prozess ist in der Subroutine TIMEA0 gekapselt und wird nur einmal ausgeführt.

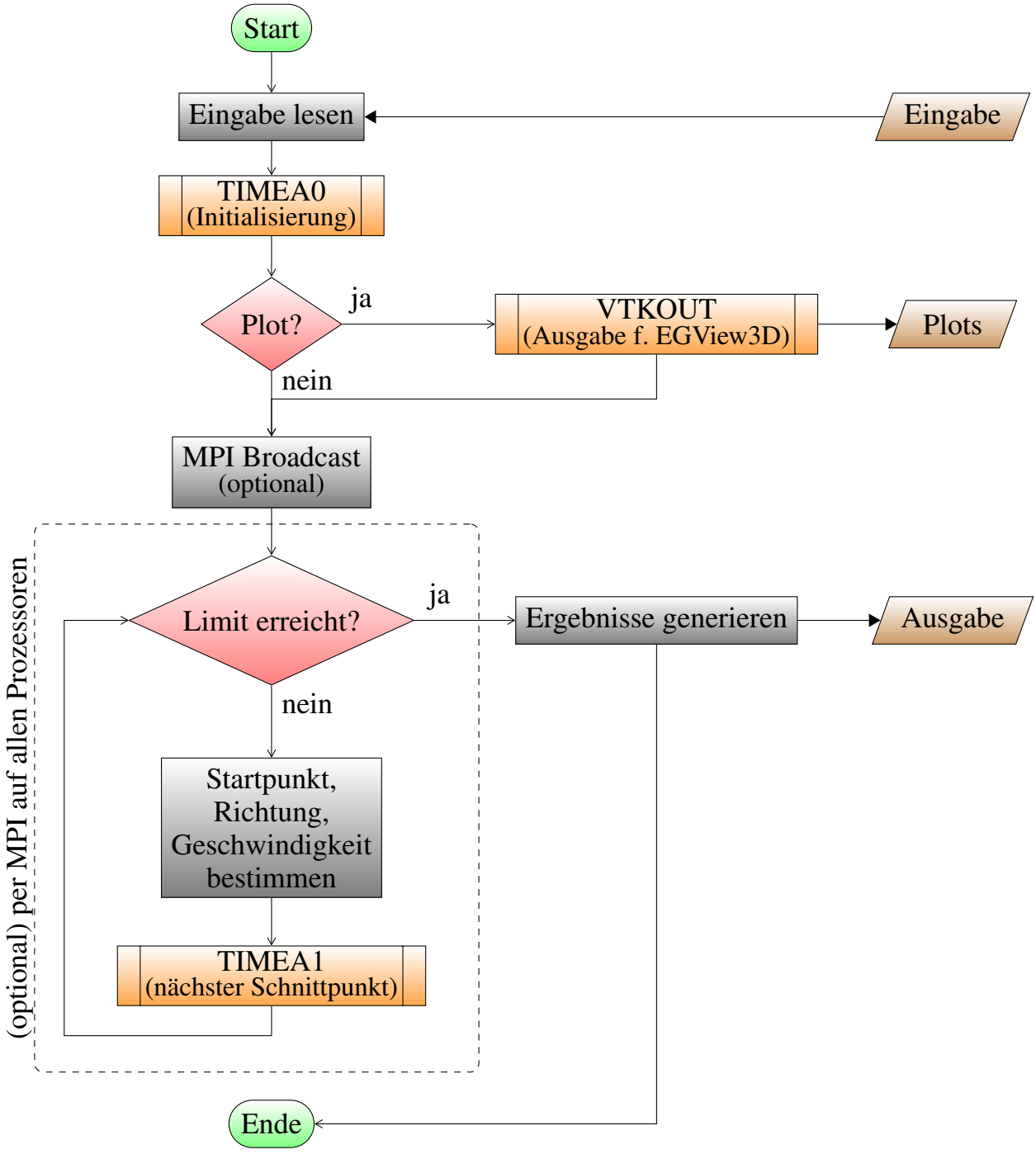


Abbildung 4.1.: Programmablauf von EIRENE mit optionaler MPI-Nutzung.
Die Grafik zeigt nur die relevanten Teile für die Simulation von „Additional Surfaces“. Darstellung ohne Abänderungen durch Octree-Implementierung. EGView3D ist ein externes Werkzeug zur Visualisierung (siehe Anhang A).

Sollte das „Message Passing Interface“ (MPI) genutzt werden, um verteilt auf mehreren Prozessoren rechnen zu können, wird TIMEA0 nur auf dem ersten Prozessor ausgeführt.

Alle Flächen, die in der Eingabe als „Additional Surfaces“ angegeben werden, sind mit einem Index versehen, der für den Zugriff auf die Daten, Koeffizienten und/oder Eckpunkte der Fläche verwendet wird. In der Subroutine TIMEA0 werden ebendiese Flächen unter Anderem auf ihre Korrektheit geprüft: Befinden sich bei Vier- oder Fünfecken nicht alle Punkte in einer Ebene oder ist die Reihenfolge der Punkte falsch - intern sind Vier- und Fünfecke in einer Ebene liegende zusammengesetzte Dreiecke - wird die entsprechende Fläche verworfen und dies dem Benutzer mitgeteilt. EIRENE erkennt anhand des Wertes einer pro Fläche definierten Fließkommazahl `RLB`, um welche Art von Fläche es sich handelt. Siehe dazu das EIRENE-Manual [2]. Zur Bearbeitung der Aufgabenstellung sind zunächst nur Flächen mit einem Wert von $3 \leq \texttt{RLB} \leq 5$ zu betrachten. Dies entspricht Drei-, Vier- und Fünfecken im Raum. Die weiteren Code-Teile in TIMEA0, die vornehmlich die Flächen 2. Ordnung betreffen, sind zu belassen, da diese wie bisher bearbeitet werden sollen. Ein Teil des Codes in TIMEA0 berechnet zu allen Flächen, sofern nicht in den Eingabedaten angegeben, die Parameter a_0 bis a_9. Diese entsprechen den Koeffizienten der impliziten Flächengleichung 2. Ordnung:

$$a_0 + a_1 x + a_2 y + a_3 z + a_4 x^2 + a_5 y^2 + a_6 z^2 + a_7 xy + a_8 xz + a_9 yz = 0$$

Für Flächen 1. Ordnung sind $a_4, \ldots, a_9 = 0$.

Nach der erfolgreichen Initialisierung wird - falls in der Eingabe aktiviert - ein Plot der Geometrie erstellt. Neben der EIRENE-internen Ausgabe einer 3D-Grafik wurde bereits im Rahmen von [3] eine XML-Ausgabe der Geometrie eingebaut, welche von dem ebenfalls in [3] grundlegend entwickelten Werkzeug „EGView3D“ genutzt wird. Dieses Programm ist eine VTK[1] basierte Java-Software zur interaktiven Darstellung von EIRENE-Geometrien und wird in Anhang A näher betrachtet.

Im Falle von Verteiltem Rechnen werden im nächsten Schritt die initialisierten Flächen vom ersten Prozessor auf die übrigen Prozessoren kopiert, indem ein MPI Broadcast für die entsprechenden Daten ausgeführt wird. Bei der optionalen Nutzung von MPI wird die tatsächliche Simulation nun auf jedem Prozessor ausgeführt, ohne MPI nur auf einem.

Die Dauer der Simulation wird im wesentlichen von zwei Parametern limitiert: Laufzeit-Maximum und Teilchen pro Quelle. Der Benutzer begrenzt die Laufzeit in den Eingabedaten auf eine Anzahl von t_{max} CPU-Sekunden, die Anzahl der Teilchen wird in der Eingabe pro Teilchenquelle vorgegeben. Ist die maximale Laufzeit abgelaufen oder sind alle Teilchen einer Quelle verfolgt worden und keine weiteren Quellen vorhanden, so werden die bis zu diesem Zeitpunkt berechneten Ergebnisse ausgegeben.

Anderenfalls wird für ein weiteres Testteilchen Startpunkt, Spezies, Richtung und Geschwindigkeit bestimmt, sodass anschließend die Teilchenverfolgung etwaige Schnittpunkte mit den Flächen der betrachteten Geometrie (1D, 2D oder 3D macht wegen der allgemeinen Form der Flächeneingabe keinen Unterschied) berechnen kann.

[1] The Visualisation ToolKit: http://www.vtk.org

Die mittels eines Monte-Carlo-Experimentes in diesem Schritt durchgeführte Berechnung der Teilchenparameter wird durch die Modellierung des Plasmas (allgemeiner: des Hintergrundmediums), der Materialeigenschaften der berandenden Flächen der Modellgeometrie etc. beeinflusst.

Als letzten Schritt in der Schleife wird der TIMEA1-Code ausgeführt. Dieser erhält die vorher bestimmten Teilchenparameter und iteriert über alle Geometrieflächen, um den nächstgelegenen Schnittpunkt zu finden. Diese lineare Suche besitzt den Aufwand $O(n)$, wie bereits in Kapitel 3.1 erläutert wurde.

Im Weiteren soll nun die Erweiterung des Codes durch die Implementierung eines Octrees vorgestellt werden. Der Abschnitt 4.2 beschreibt die Erzeugung des Baumes, während der Abschnitt 4.3 die abgewandelte Form der Routine `TIMEA1` darlegt.

4.2. Aufbau des Baumes

Die Erweiterung des Codes sollte auf den bisherigen Schnittstellen aufsetzen, um die grundlegende Struktur der bisherigen Software weitestgehend zu erhalten. Ebenso sollten die bisherigen Codeteile zur eigentlichen Schnittpunktberechnung möglichst erhalten bleiben. (vgl. Kapitel 2). Die zur Beschleunigung in Kapitel 3.2.3 identifizierte Strategie der raumteilenden hierarchischen Datenstruktur der Octrees ist demnach in der Initialisierungsroutine `TIMEA0` anzuwenden und dort ist ein entsprechender Octree aufzubauen (siehe Abbildung 4.1 und 4.2). Der aus dem Raytracing bekannte Traversal-Schritt wird Teil des `TIMEA1`-Codes, da dort die Teilchen-Strahlverfolgung ausgeführt wird.

Der Ablauf der Routine `TIMEA0` wird zu dem in Abbildung 4.2 dargestellten abgewandelt. Da es dem Benutzer ermöglicht werden soll, den alten Code gänzlich ohne Octree wie

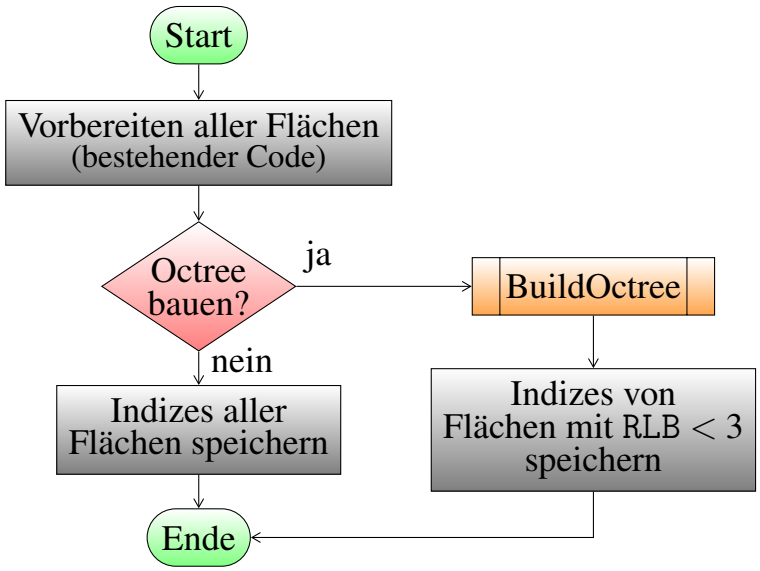


Abbildung 4.2.: Ablaufdiagramm für die Routine `TIMEA0`.

bisher nutzen zu können, wird nur im Falle der Aktivierung der Erweiterung in der Eingabe der Baum aufgebaut. In beiden Fällen wird das Feld `notOCSurfs` mit Indizes von Flächen gefüllt. Wird ein Octree benutzt, werden darin die Flächenindizes aller nicht im Octree berücksichtigten Additional Surfaces hinterlegt. Ohne Baum werden alle Additional Surfaces darin gespeichert. Der Grund für diese Lookup-Table liegt in der in Abschnitt 4.3 näher erläuterten Umstrukturierung des TIMEA1-Codes.

Alle „Additional Surfaces“ sind von anderen in EIRENE möglichen Flächen unabhängig, sodass nur diese für den Aufbau des Octrees von Bedeutung sind. Bei der Erstellung des Baumes werden ferner aus der Menge der Add. Surfaces nur Drei-, Vier- und Fünfecke ($3 \leq$ `RLB` ≤ 5, s. o.) berücksichtigt. Die große Mehrzahl aller im Institut vorhandenen Geometrien weisen keine Flächen 2. Ordnung auf und die, die diese Flächen nutzen, setzen nur eine kleine zweistellige Anzahl davon ein. Die Geometrien, die bereits für das ITER-Projekt simuliert wurden, weisen durch die Triangulierung der durch die CAD-Programme erzeugten Flächen bis fünfter Ordnung keine anderen Flächentypen als ausschließlich Dreiecke auf. Es ist somit für den realen Anwendungsfall zunächst hinreichend, lediglich begrenzte Flächen 1. Ordnung zu berücksichtigen.

Grenzfälle von Flächen 1. Ordnung, die Löcher in anderen Flächen definieren, werden in der Implementierung erst nachfolgend berücksichtigt. Zum Zeitpunkt der Erstellung dieser Arbeit waren jedoch keine Geometrien bekannt, die solche Flächenkonstrukte nutzen, sodass dieser Aspekt als zweitrangig betrachtet wurde und ggf. Teil der zukünftigen Weiterentwicklung sein wird.

Zur Speicherung des Baumes wird eine Datenstruktur mit baumweiten Eigenschaften und einem Zeiger auf den Wurzelknoten benötigt. Die Definition von „Derived Types“ in Fortran ermöglicht die Erstellung eines solchen verschiedene Daten kapselnden Datentyps. Im neu erstellten Modul `EIRMOD_OCTREE` erfolgt die Definition eines „Derived Type“ `ocTree`, dessen Struktur in Tabelle B.1 im Anhang aufgezeigt wird. Für die einzelnen Knoten des Baumes wird ebenso eine adäquate Datenstruktur benötigt. Dabei soll durch Zeiger auf die acht Kinder und einen Zeiger auf das Elternelement der eigentliche Aufbau des Baumes erfolgen. Tabelle B.2 im Anhang zeigt den Aufbau des Datentyps `ocNode`.

Abbildung 4.3 zeigt das Ablaufdiagramm der Erstellung des Octrees. Die in `TIMEA0` gerufene Routine `BuildOctree` alloziert für den Baum eine Variable `tree` vom Typ `ocTree` und erzeugt den Wurzelknoten `root` vom Typ `ocNode`. In Abschnitt 4.2.1 wird die Bildung der konvexen Hülle sowie das Finden der Parameter des Baumes beschrieben. Alle Flächen mit $3 \leq$ `RLB` ≤ 5 aus `TIMEA0` werden mit dem Wurzelknoten durch Eintragen der Indizes in `surfaces` assoziiert.

Die in `BuildOctree` für eine Baumhöhe $h > 1$ für den Wurzelknoten gerufene Routine `BuildBlocks` unterteilt das Volumen mit der Routine `CreateChildren`. Die Funktionsweise dieser Routine wird in Abschnitt 4.2.2 beschrieben.

Alle acht Kinder werden sequentiell bearbeitet. Die Algorithmen, die die Menge der Flächen assoziieren, die in den jeweiligen acht Kindern, Kindeskindern, ... der Wurzel liegen, werden in Abschnitt 4.2.4 vorgestellt.

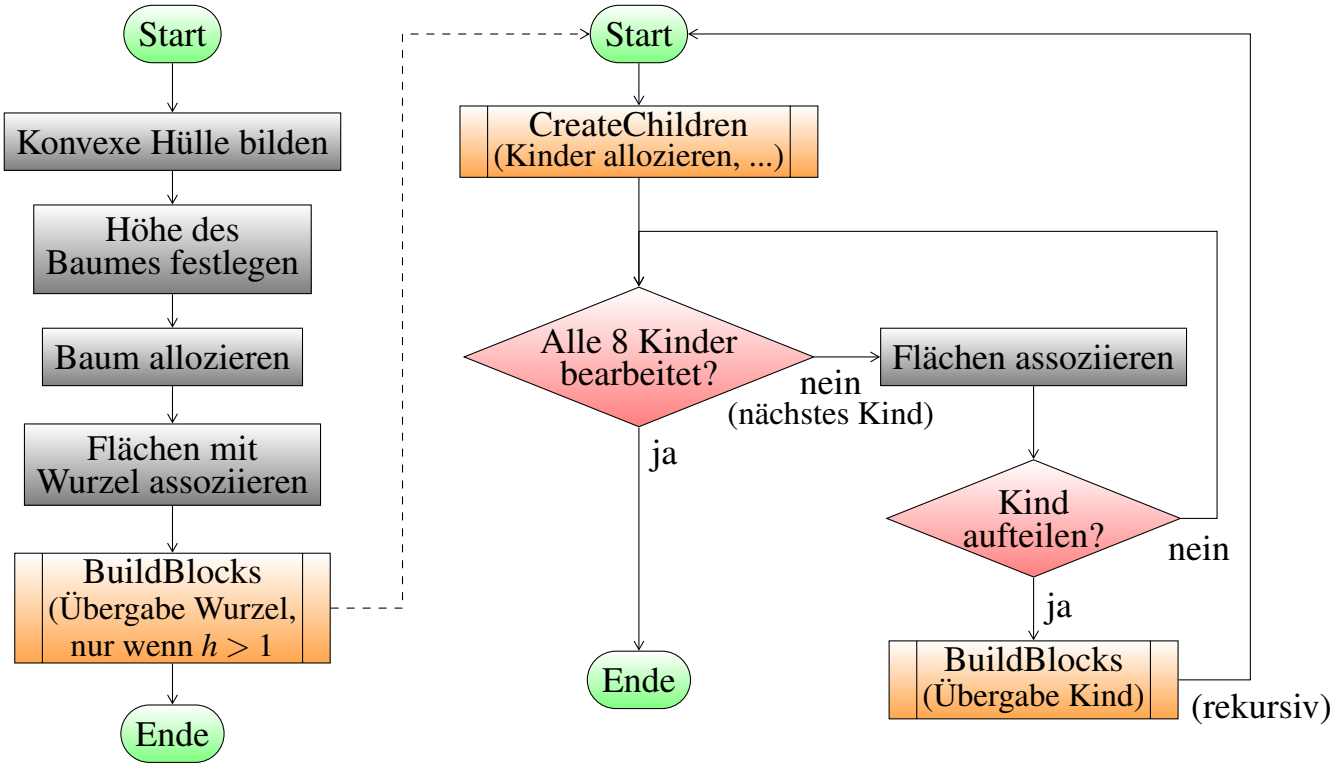


Abbildung 4.3.: Ablaufdiagramm für Octree-Aufbau (Abb. 4.2, BuildOctree).
Links: BuildOctree-Routine; Rechts: BuildBlocks-Routine, gerufen von BuildOctree bzw. rekursiver Selbstaufruf

Liegen in einem Kind mehr Flächen als die Anzahl, die die in Abschnitt 4.2.1 definierte obere Schranke a zulässt und ist die ebenfalls dort definierte maximale Höhe h des Baumes noch nicht erreicht, wird das Volumen des Kindes durch rekursiven Aufruf der Routine `BuildBlocks` mit dem Kindknoten als Parameter weiter unterteilt.

Der so erzeugte Baum wird für den Gebrauch in der Routine `TIMEA1` gespeichert, indem ein Zeiger auf `root` in `tree` gespeichert wird.

4.2.1. Parameter des Baumes

Um ein Volumen zu generieren, das weiter unterteilt werden kann und in dem alle relevanten Flächen eingeschlossen sind, muss die konvexe Hülle aller zu berücksichtigen Flächen erstellt werden.

Diese ist ein Boundary Volume in Form eines Quaders, der durch den unteren linken Eckpunkt B_1 und den oberen rechten Eckpunkt B_2 beschrieben ist. Beide Punkte ergeben sich aus der Minima- bzw. Maxima-Bildung der Komponenten aller m Eckpunkte aller n berücksichtigen Flächen $(m \geq 3 * n)$:

$$B_1 = \begin{pmatrix} \min\limits_{i=1..m}(x_i) \\ \min\limits_{i=1..m}(y_i) \\ \min\limits_{i=1..m}(z_i) \end{pmatrix}, \qquad B_2 = \begin{pmatrix} \max\limits_{i=1..m}(x_i) \\ \max\limits_{i=1..m}(y_i) \\ \max\limits_{i=1..m}(z_i) \end{pmatrix}$$

Abbildung 4.4 zeigt beispielhaft das Vorgehen. Die Hülle entspricht der Wurzel des Octrees, die den gesamten Teilraum repräsentiert, auf dem der Baum definiert ist.

Um spätere Probleme mit Fließkomma-Arithmetik bei der Untersuchung von Zugehörigkeit von beliebigen Punkten mittels „Location Codes“ (siehe Abschnitt 4.3.1) zu vermeiden, wird die konvexe Hülle um einen kleinen Betrag ausgeweitet, in Abbildung 4.4 gestrichelt dargestellt:

$$\tilde{B}_1 = B_1 - 10^{-12} * \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \qquad \tilde{B}_2 = B_2 + 10^{-12} * \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Zur Erstellung des Wurzelknotens werden B_1 und B_2 genutzt, um die Unterteilungen an den ursprünglichen Grenzen auszurichten. Die Werte von $\tilde{B}_1$ und $\tilde{B}_2$ werden separat in einer Variablen des Datentyps `ocTree` gespeichert. Des Weiteren wird im Baum die Länge der Kanten des erweiterten Boundary Volumes gespeichert: $length = \tilde{B}_2 - \tilde{B}_1$.

Nach der Erstellung des Wurzelknotens kann der umschlossene Raum weiter unterteilt werden. Der dazu implementierte Algorithmus soll die Unterteilungen möglichst adaptiv zur Geometrie anlegen, sodass die in Abschnitt 3.2.3 erwähnte Imbalance nicht oder nur in geringem Maße auftritt.

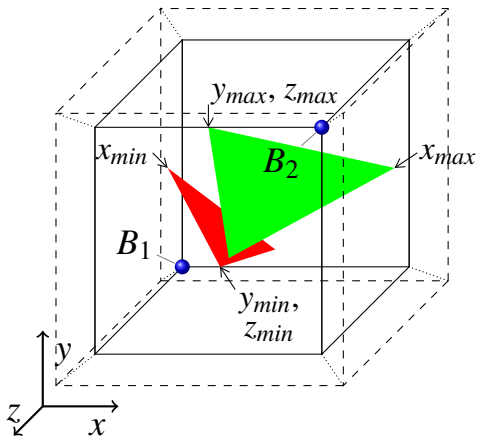


Abbildung 4.4.: Erstellung der konvexen Hülle als Volumen für den Octree.
Die gestrichelte Hülle entspricht der um 10^{-12} erweiterten Hülle zur Vermeidung von Problemen mit der Fließkomma-Arithmetik.

Anzahl Flächen	max. Höhe des Baumes
$\leq 10^1$	1
$\leq 10^2$	3
$\leq 10^3$	4
$\leq 10^4$	5
...	

Tabelle 4.1.: Höhe des Baumes im Verhältnis zur Flächenzahl im Octree.
Daten berechnet nach Formel 4.1, für max. 10 Flächen bereits adaptive Unterteilung berücksichtigt.

Das erste dynamische Kriterium zur Steuerung der Unterteilung betrifft die Höhe des Baumes. Das Ziel ist, in einem Blattknoten eine möglichst geringe Anzahl von Flächen anzusiedeln, die durch den TIMEA1-Code getestet werden müssen. Parallel sollte der Baum eine minimale Anzahl an Ebenen aufweisen, um die Suche nach Knoten in wenigen Rekursionsschritten bewältigen zu können. Aus diesem Grund orientiert sich in der Implementierung die Höhe h des Baumes an der Anzahl n der Flächen, die in ihm gespeichert werden sollen:

$$h = \lfloor log_{10}(n) \rfloor + 1 \tag{4.1}$$

Die Werte der Tabelle 4.1 zeigen, dass die Anzahl der Ebenen durch den 10er-Logarithmus der Flächenanzahl bei größer werdenden Geometrien nur langsam ansteigt.

Der spätere Algorithmus zur Suche nach einem Knoten ist also bereits durch den Aufbau des Baumes nach oben in der Rekursionstiefe beschränkt. Aus dieser Beschränkung lässt sich keine Aussage über die Balance eines Baumes ableiten. Eine Entartung des Baumes mit einem vielfachen der hier angegebenen Höhe ist jedoch nicht möglich.

Ein weiteres, die Höhe des Baumes beeinflussendes Kriterium sorgt für die adaptive Anpassung an die Geometrie: Eine obere Schranke a bewirkt, dass Knoten für Bereiche mit wenigen Flächen keine Subebenen erzeugen. In der Implementierung wurde empirisch festgelegt, dass ein Octet nicht weiter unterteilt wird, wenn es weniger als 11 Flächen beinhaltet. Die Höhe des Baumes mit maximal zehn Flächen ist daher in Tabelle 4.1 nicht nach Formel 4.1 angegeben, sondern berücksichtigt bereits die Nichtunterteilung des Wurzelknotens.

4.2.2. Unterteilung des Baumes

Ausgehend vom Wurzelknoten wird der Octree durch rekursive Unterteilung von Octeten aufgebaut. Die Unterteilung folgt dabei den in Abschnitt 4.2.1 definierten Parametern. Sind in einem Octet mehr als a Flächen, wird das Octet weiter unterteilt.

Das blockförmige Octet wird in acht größengleiche Blöcke aufgetrennt. Die Blöcke werden durch die Punkte B_1, B_2 und B_3 bestimmt. B_1 und B_2 sind aus dem Elternknoten im Baum bekannt, der durch die Teilung Kinder erhalten soll. Der Raummittelpunkt B_3

des Octets generiert sich aus diesen Punkten:

$$B_3 = B_1 + 0,5 * (B_2 - B_1) \tag{4.2}$$

Durch die Berechnung von B_1 und B_2 des Wurzelknotens als Minimum bzw. Maximum gilt im Wurzelknoten $B_{1i} \leq B_{2i}$ für jede Komponente: $i = x, y, z$. Für jede weitere Unterteilung anhand der Raummitte in B_3 bleibt diese Eigenschaft erhalten. Die Gleichung 4.2 ist ohne Beschränkung der Allgemeinheit anwendbar.

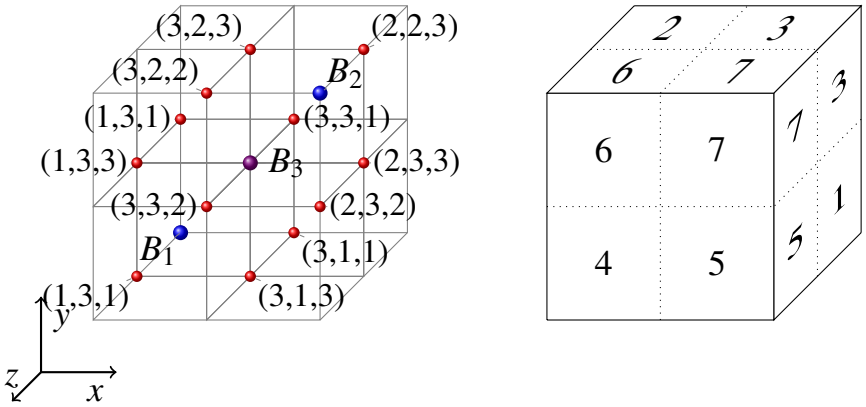


Abbildung 4.5.: Generierung der Eckpunkte B'_1, B'_2 aller Kinder aus B_i, $i = 1, ..., 3$
Rote Punkte: x_j, y_k, z_l als (j,k,l) geschrieben.
rechts: Octet-Indizes für links, verdecktes Octet hat Index 0.

Die acht Kinder erhalten ihre B'_1 und B'_2 als Kombinationen der in Abbildung 4.5 und Tabelle 4.2 beschriebenen Komponenten von B_1, B_2 und B_3. Zu einem Knoten gehört eine Matrix B der Form

$$\underbrace{\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{pmatrix}}_{B_i}$$

in welcher die B_i angelegt wird.

$$B'_i = \begin{pmatrix} B_j x \\ B_k y \\ B_l z \end{pmatrix}, i = 1,2 \qquad j,k,l \in \{1,2,3\}$$

und j,k,l aus der Tabelle 4.2 generiert die beiden Eckpunkte B'_1, B'_2 für jedes Kind.

Octet-Index	0	1	2	3	4	5	6	7
X-Richtung	j-Indizes							
B'_1	1	3	1	3	1	3	1	3
B'_2	3	2	3	2	3	2	3	2
Y-Richtung	l-Indizes							
B'_1	1	1	3	3	1	1	3	3
B'_2	3	3	2	2	3	3	2	2
Z-Richtung	k-Indizes							
B'_1	1	1	1	1	3	3	3	3
B'_2	3	3	3	3	2	2	2	2

Tabelle 4.2.: Generierung der Eckpunkte der Kinder. $B'_i = \begin{pmatrix} B_j x \\ B_k y \\ B_l z \end{pmatrix}, i = 1,2$

Im Code werden die Indizes j, k und l aus statisch angelegten Lookup-Tables gelesen, die den Tabelle 4.2 gleichen Inhalt aufweisen. Für die drei Raumdimensionen existieren getrennte Tabellen, sodass insgesamt drei Matrizen der Größe 8x2 im Speicher liegen.

Die Ebenen des Octrees werden von oben nach unten absteigend numeriert. Der Wurzelknoten hat als Ebene die Nummer `layer`$= h-1$. Alle Kinder dekrementieren diese Ziffer, sodass die äußersten Blätter die Ebene 0 haben. Blätter, die durch Nichtunterteilung von Knoten auf höheren Ebenen entstehen, tragen auch die Ebenennummer dieser höheren Ebene.

Bei der Erzeugung der Kinder in der Routine `CreateChildren` wird zunächst die Ebene des Baumes `layer` bestimmt, in der die Kinder liegen, indem die Ebenennummer des Elternknotens als dekrementierte Kopie gespeichert wird. Nach der Allokation der Kindknoten werden sequentiell allen Kindern 0,...,7 folgende Eigenschaften gegeben:

1. Zuweisung des Octet-Volumens durch B'_1 und B'_2
2. Zuweisung des Location Codes, siehe nächster Abschnitt
3. Zeiger auf Elternknoten
4. Generierung eines neuen B_3
5. Allokation des Feldes `surfaces` mit der Anzahl `nsurfaces` des Elternelementes
6. Berechnung des Radius der Umkugel mit $r = 0,5 * ||B'_2 - B'_1||_2$
7. Berechnung der kürzesten Kante
   a) ggf. Eintragung in `tree`, siehe 4.3.3

Die Kinder werden im Elternknoten referenziert, indem sequentiell in einem achtelementigen Feld `children` Zeiger auf das entsprechende Kind hinterlegt werden. Ist ein Elternknoten ein Blatt, so ist dieses Feld nicht alloziert. Alle Kinder erhalten einen Zeiger auf das Elternelement, um eine Bottom-Up-Nachbarsuche zu ermöglichen (siehe Abschnitt 4.3.3).

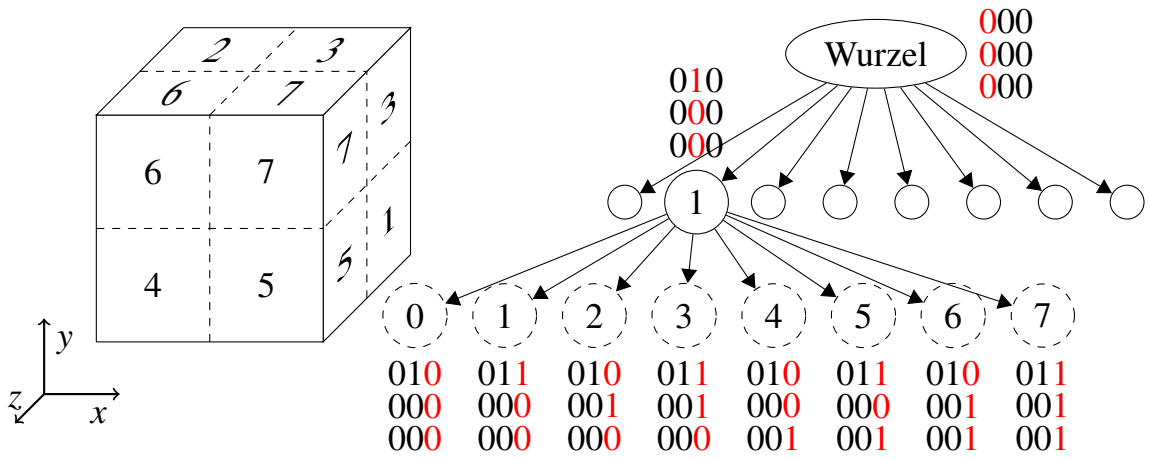


Abbildung 4.6.: Location-Code (LC) Beispiel für einen Baum mit $h = 3$.
Der LC pro Knoten ist unabhängig für die drei Raumrichtungen (x, y, z) in der Beschriftung untereinander aufgeführt. Die roten Zahlen markieren die zur Baumebene gehörigen Ziffern. Der Block entspricht dem Knoten auf der zweiten Ebene, die Unterblöcke den Blättern.

4.2.3. Location Codes

Basierend auf den Ausführungen von [1] wird die Nachbar- und Punktsuche mit Hilfe von sogenannten „Location Codes“ beschleunigt. Die Suche ist iterativ möglich, sodass keine Rekursionsstufen notwendig sind, die bei der erwarteten hohen Anzahl an Knotensuchen einen potentiellen Flaschenhals darstellen.

Die grundlegende Idee ist, dass ein Knoten für jede Dimension eine unabhängige Binärzahl zugewiesen bekommt. Die Binärzahl hat ebenso viele Stellen wie der Baum Ebenen hat.

Die einzelnen Stellen werden von links nach rechts gesetzt und gelesen, d. h. die Wurzelebene entspricht der vordersten Binärziffer, die letzte Binärziffer der tiefsten Ebene des Baumes.

Ein Octetvolumen wird immer genau an seinem Mittelpunkt aufgetrennt (siehe Abschnitt 4.2.2). Die Binärziffer 0 in einem Location Code drückt aus, dass das Octet, zu dem der Code gehört, das in der jeweiligen Raumrichtung vor diesem Punkt im Elternvolumen liegt. Die Binärziffer 1 drückt aus, dass es hinter dem Punkt liegt. Abbildung 4.6 zeigt beispielhaft die Zuordnung von Location Codes bei einem Ausschnitt eines Octrees.

Bei der Erzeugung von Kindern auf der Ebene e in einem Knoten des Octrees, der auf Ebene $e+1$ liegt, erhalten die Kinder ihren Location Code für die einzelnen Raumrichtungen auf Basis des Location Codes des Elternknotens. Dabei wird anhand dessen, ob sie vor oder hinter B_3 liegen, die e-te Ziffer der Binärzahl auf 1 oder 0 gesetzt.

Beispiel: der Location Code des Elternknotens aller Blätter in Abbildung 4.6 lautet $LC_x, LC_y, LC_z = (010)_2, (000)_2, (000)_2$. Der Kindknoten (3) erhält als Location Code in x-Richtung $(011)_2$, in y-Richtung $(001)_2$ und in z-Richtung $(000)_2$. Diese Kombination erklärt sich daraus, dass (3) in x- und y-Richtung hinter B_3 liegt, in z-Richtung jedoch vor B_3 und die entsprechenden Bits der Binärzahl angeschaltet werden.

Bei der Bearbeitung der acht Kinder in der Routine `CreateChildren` werden anhand der bereits im vorherigen Abschnitt genutzten Lookup-Tables für die Generierung der Kindknoten auch die Bits der Binärzahl gesetzt. Ist ein Index für B'_1 in Tabelle 4.2 gleich 3, liegt das Octet in der entsprechenden Raumrichtung hinter dem Punkt B_3. Für diese Raumrichtung wird das Bit des LC an der Stelle e gesetzt. Bei einer 1 oder 2 wird das Bit auf $(0)_2$ belassen. Die Position e des Bits entspricht der Ebene (s. o.), auf der der Kindknoten liegt und bereits bei der Generierung des Knotens als Variable `layer` in der Datenstruktur hinterlegt wurde.

Die ohne Begründung in Abschnitt 4.2.2 geforderte Nummerierung der Ebenen im Baum absteigend von $(h-1)$ bis 0 ist notwendig, da das Setzen der Bits von links - höchster Index der Stelle $2^{(h-1)}$ - nach rechts - Index der Stelle 2^0 - erfolgen soll, um das „least significant bit“ (LSB) des Wurzelknotens an der linkesten Position zu setzen.

Die Verwendung der Location Codes wird in Abschnitt 4.3.1 beschrieben.

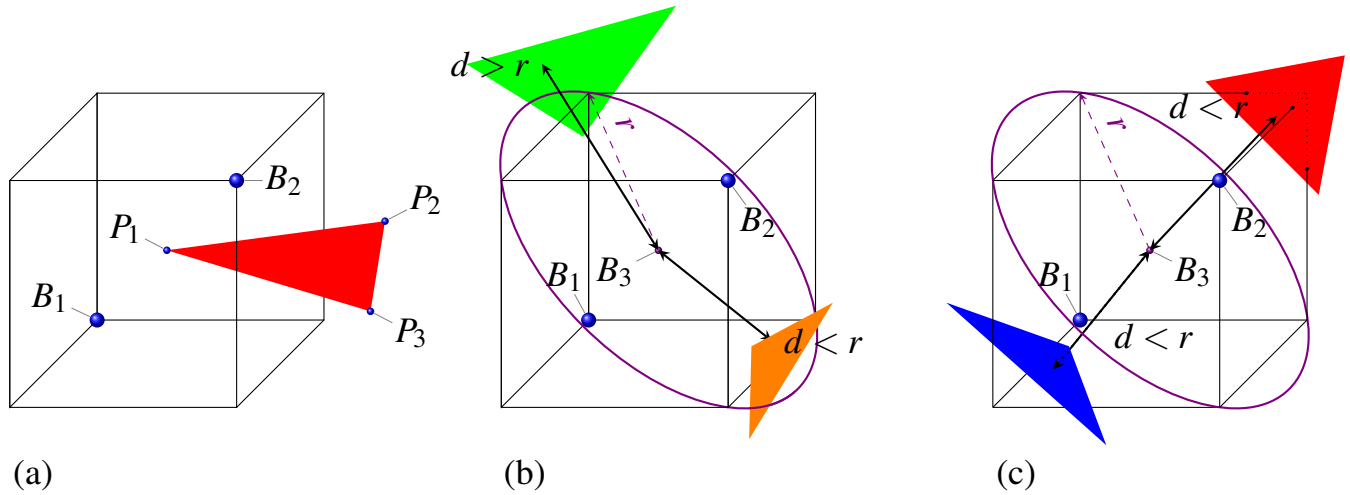
4.2.4. Assoziation von Flächen

Parallel zur Erstellung eines Knotens müssen mit diesem alle Flächen assoziiert werden, die ganz oder teilweise in diesem liegen. Gesucht sind dazu möglichst unaufwändig durchführbare Tests, die die Zugehörigkeit einer Fläche zu einem Octet prüfen.

Alle Flächen, die im Octree assoziiert werden sollen ($3 \leq$`RLB`≤ 5, s. Abs. 4.2), werden zunächst dem Wurzelknoten - der konvexen Hülle - zugeordnet. Während das Volumen auf Kindknoten aufgeteilt wird, werden auch die im Volumen gelegenen Flächen mit den Kindknoten assoziiert, falls die Fläche ganz oder teilweise in diesem Unterraum liegt.

Zur Referenzierung der Daten der Fläche (Eckpunkte, Vektoren, etc.) ist es hinreichend, den während eines EIRENE-Laufes codeweit eindeutigen Index der Fläche im Knoten zu speichern. Die Indizes werden mit Hilfe eines Feldes `surfaces` pro Knoten gespeichert (siehe Datenstruktur in Tabelle B.2). Die in `surfaces` gespeicherte Menge der Flächenindizes ist nicht disjunkt zu den Mengen anderer Octete; ragt eine Fläche über ein Octet hinaus, wird sie in mehreren Octets assoziiert.

Durch Hinterlegen der Indizes der Flächen im Feld `surfaces` des Wurzelknotens werden alle Flächen diesem zugeordnet. In jedem Raumvolumen eines Kindknotens kann maximal dieselbe Menge Flächen liegen, die im Volumen des Elternknotens liegen. Die Feldgröße zur Speicherung der Indizes im neuen Kindknoten wird anhand dieser Anzahl begrenzt und bei Erstellung des Kindknotens zur Speicherallokation genutzt.



(a) (b) (c)

Abbildung 4.7.: Zuordnung von Flächen zu einem Octet.
Umkugel als Äquator in violett angedeutet.
(a) min. ein Punkt im Octet (hier P_1)
(b) Abstand orange Fläche $d < r$, schneidet Umkugel, liegt nicht in Octet
(c) Abstand Flächen $d < r$, Durchstoßtests positiv.

Um festzustellen, welche Kindknoten welche Flächen assoziieren müssen, werden bis zu drei Prüfungen mit jeder Fläche des Elternknotens absolviert:

1. Liegt mindestens ein Eckpunkt der Fläche im Volumen des Kindknotens?
2. Ist der Abstand der Fläche zur Raummitte des Octets kleiner als der Radius der Umkugel? (≙Abstand Punkt-Ebene)
3. Durchstoßprüfung:
   a) Durchstößt mindestens eine Kante des Octets die Fläche?
   b) Durchstößt mindestens eine Kante der Fläche das Octet?

In Abbildung 4.7 werden diese Tests exemplarisch dargestellt. Test Nr. 1 ist positiv für die Fläche in Abb. 4.7(a), da der Punkt P_1 innerhalb des Octetvolumens liegt.

Flächen, für die dies nicht zutrifft, werden anhand Test Nr. 2 näher beurteilt. Die Prüfung des Abstandes erfolgt über die Berechnung des Abstandes des Raummittelpunktes B_3 zur Ebene, in der die Fläche liegt:

$$d = \frac{1}{\sqrt{(a_1,a_2,a_3) * \overrightarrow{B_3}}} * \left| a_0 + (a_1,a_2,a_3) * \overrightarrow{B_3} \right|$$

Ist der Abstand $d < r$, wobei $r = 0,5 * ||B_2 - B_1||_2$ der Radius der Umkugel des Octets ist, wird die Fläche näher untersucht. Der Radius wurde bereits bei Anlegen des Knotens berechnet (siehe Abschnitt 4.2.2).

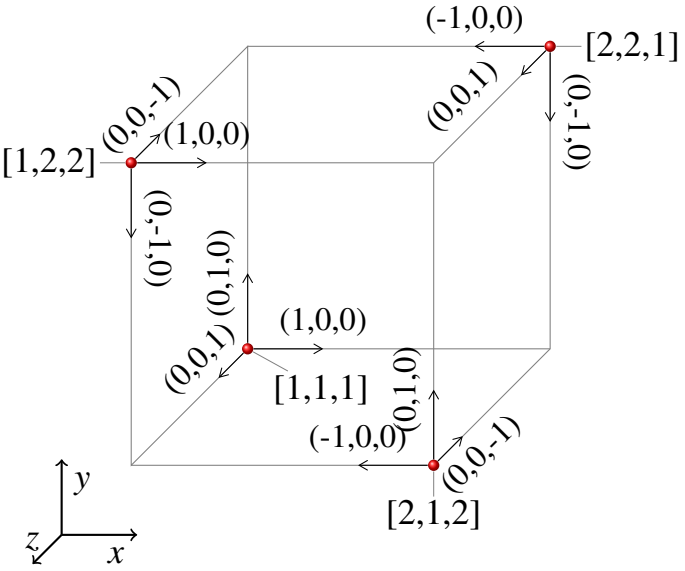


Abbildung 4.8.: Testgeraden aus den Kanten des Octets.
Angabe $[j,k,l]$ an den roten Eckpunkten beschreibt Ortsvektoren $O = (B_j x, B_k y, B_l z)^T$.
Kanten entsprechen den Geraden mit angegebenen Richtungsvektoren (x,y,z).

Alle Flächen, die einen Abstand $d > r$ aufweisen, können unter keinen Umständen innerhalb des Octetvolumens liegen, da die Ebene das Octet nicht schneiden kann, weil sie bereits die Umkugel nicht schneidet. Abbildung 4.7(b) zeigt in grün einen solchen Fall.

Dieselbe Abbildung 4.7(b) zeigt mit der orangen Fläche jedoch auch, dass dieses Kriterium die Aussagen $d < r$ und Fläche in Octet nicht äquivalent sind: Der leere Raum um die Kugel, die ein Boundary Volume für den quaderförmigen Block des Octetvolumens darstellt, kann von Ebenen geschnitten werden, ohne dass die Ebene das eigentliche Volumen schneidet.

Für den Fall $d < r$ ist mit einem dritten Test auf Durchstoß zu prüfen. Dabei kann eine Kante des Volumens die Fläche durchstoßen oder eine Kante der Fläche das Volumen. Abbildung 4.7(c) zeigt beide Fälle, in denen kein Eckpunkt der Fläche innerhalb des Volumens liegt, die Fläche aber dennoch mit dem Octet assoziiert werden muss.

Zunächst werden alle zwölf Kanten des Octets als Geraden in Parameterdarstellung aufgefasst:

$$g : x = O_i + \lambda d_i,\ i = 0, \ldots, 11$$

Dabei wird der Ortsvektor O_i durch einen Eckpunkt des Octets definiert. Die Kanten aller Octete sind achsenparallel, entsprechen damit den Elementarvektoren und bilden die Richtungsvektoren d.

Die Ortsvektoren werden ähnlich der Generierung von B'_1 und B'_2 mit Hilfe einer Lookup-Table mit Indizes generiert. Die in der Tabelle enthaltenen Indizes j, k, und l für die vier Ortsvektoren sind in Abbildung 4.8 in der Form $[j,k,l]$ aufgetragen.

Dabei setzt sich jeder Vektor O_i zusammen aus $O_i = (B_j x, B_k y, B_l z)^T$. Zu jedem Eckpunkt gehören drei Richtungsvektoren d_i, somit werden für zwölf Geraden vier Ortsvektoren und zwölf Richtungsvektoren benötigt.

Die drei Komponenten der Richtungsvektoren sind bis auf eine gleich 0. Diese Einzelkomponente wird in einer weiteren Lookup-Table gespeichert. Dabei liegen die Werte wie folgt vor: $dvec = (x_0, y_1, z_2, x_3, y_4, z_5, x_6, y_7, z_8, x_9, y_{10}, z_{11})$. Der Vektor d_i ist als Feld mit $(0,0,0)$ zu initialisieren und mit $d_i(mod(i,3)+1) = dvec(i)$ zum gewünschten Richtungsvektor abzubilden, wobei $d_i(pos)$ das Element an Index pos im Feld d_i meint.

Der Schnittpunkttest der Geraden mit den Flächen erfolgt unter Ermittlung der baryzentrischen Koordinaten (siehe Abschnitt 3.3). Erfüllt die Lösung des Gleichungssystems die Anforderung an die baryzentrischen Koordinaten ($\beta + \gamma < 1$, $0 < \beta, \gamma < 1$,$\lambda > 0$), ist die Lösung (β, γ, λ) gültig. Die Gerade trifft die Ebene der Fläche im Schnittpunkt $S = O_i + \lambda d_i$ innerhalb des durch die Eckpunkte definierten Ausschnitts. Für Dreiecke wird ein einzelner Test benötigt. Vier- und Fünfecke bestehen aus zwei bzw. drei Dreiecken in derselben Ebene, somit muss der Test ein bzw. zweimal mit den übrigen Dreiecken wiederholt werden.

Wird mindestens eine gültige Lösung gefunden, wird die Fläche mit dem Octet assoziiert. Findet sich keine solche, können die Kanten der Fläche das Octet durchstoßen, ohne dass die Kanten des Octets die Fläche treffen (siehe blaue Fläche in Abbildung 4.7(c)). Entsprechend dem vorherigen Test werden Vier- und Fünfecke in ihre einzelnen Dreiecke zerlegt und getestet.

Jede Kante der Dreiecksfläche wird als Gerade aufgefasst. Die Seitenflächen des Octets bilden Ebenen, mit denen die Geraden auf Schnittpunkte geprüft werden. Liegt mindestens ein Schnittpunkt innerhalb des Octetvolumens, durchstößt die Fläche das Octet.

Die Kanten eines Dreiecks werden mittels der Eckpunkte definiert durch

1. $g_1 : x = P_1 + \lambda(P_2 - P_1)$
2. $g_2 : x = P_2 + \lambda(P_3 - P_2)$
3. $g_3 : x = P_3 + \lambda(P_1 - P_3)$.

Die sechs Ebenen der Seiten des Octetblocks sind als Hessesche Normalform

$$E : n * (x,y,z)^T - n * B_i = 0$$

gegeben. Die Normalenvektoren, die den Elementarvektoren entsprechen, sind in das Innere des Volumens gerichtet. B_i bezeichnet den Aufpunkt der Seitenfläche und entspricht dabei einem der beiden Eckpunkte B_1 oder B_2 des Octets. Abbildung 4.9 zeigt die Normalenvektoren und die beiden Aufpunkte der Ebenen.

Die Routine `CheckBlock` des Modules `EIRMOD_OCTREE` führt die nominell sechs Schnittpunkttests für jeweils eine der drei Geraden $x = O + \lambda d$ aus. Um die Zahl der Tests zu verringern, wird die Gerade nur mit den Ebenen geschnitten, deren Skalarprodukt von Richtungsvektor d und Normalenvektor n mit $d * n > 0$ ist, wie [20] ohne nähere Erläuterung angibt, die im folgenden aufgezeigt wird.

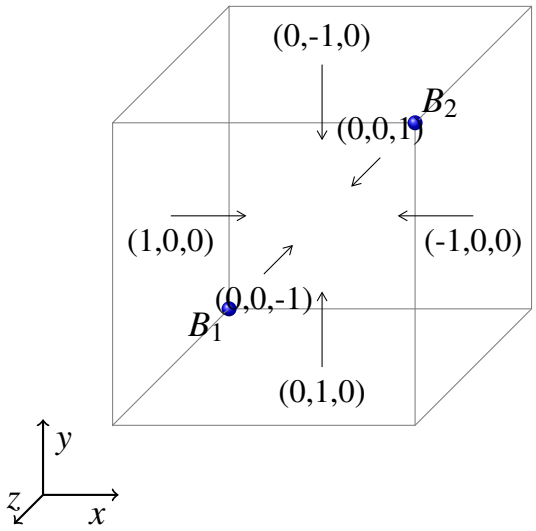


Abbildung 4.9.: Seitenflächen des Octets als Ebenen.
Ebenen als HNF: $n * (x,y,z)^T - n * B_i = 0$. B_i entspricht einem blauem Eckpunkt.

Die Idee ist, nur die Ebenen zu testen, die vom Aufpunkt der Gerade aus „sichtbar“ sind. Dies ist z. B. für die Oberseite eines Blockes nur dann der Fall, wenn die Gerade in z-Richtung von positiv nach negativ gerichtet ist und der Aufpunkt oberhalb der Ebene der Oberseite liegt. Für die „sichtbaren“ Flächen muss gelten $d * n > 0$, d. h. zwischen den beiden Vektoren befindet sich ein Winkel $0° \leq \alpha < 90°$. Dies gilt, weil die Normalenvektoren der Seitenflächen in das Innere des Octets zeigen und der Richtungsvektor der Geraden nicht in die entgegen gesetzte Richtung zeigen darf, das aber bei $d * n \leq 0 \Rightarrow 90° \leq \alpha \leq 180°$ der Fall wäre.

Die Flächen, für die $d * n < 0$ gilt, werden von den anderen Flächen verdeckt. Im Normalfall weisen drei Ebenen die Eigenschaft $d * n > 0$ auf, falls die Gerade parallel zu einer Fläche liegt ($d * n = 0$) nur zwei. Die Anzahl der Schnittpunkttests verringert sich somit von sechs auf durchschnittlich drei.

Die Berechnung des Schnittpunktes ist ohne baryzentrische Koordinaten lösbar. Zunächst wird der Parameter

$$\lambda = \frac{n * B_i - n * O}{n * d}$$

für alle drei Ebenen berechnet. Durch Einsetzen von λ in die Geradengleichung wird der eigentliche Schnittpunkt S berechnet. Gilt komponentenweise $B_1 \leq S \leq B_2$ und $0 < \lambda < 1$, so liegt der Schnittpunkt innerhalb des Octets und auf der Gerade zwischen zwei Eckpunkten. Es muss gelten $\lambda > 0$, da ein Teilchen nicht „rückwärts“ fliegen kann. Die Kante durchstößt somit das Volumen des Octets und die Fläche wird mit dem Octet assoziiert. Es ist hinreichend, eine Lösung zu finden. Alle weiteren Kombinationen müssen nicht untersucht werden.

4.2.5. Verwendung von verteiltem Rechnen

Der EIRENE-Code kann optional auf mehreren Prozessoren rechnen. Dazu wird das Message-Passing-Interface (MPI) verwendet.

Die Initialisierung der Geometrie mittels der Routine `TIMEA0` wird nur auf einem Prozessor durchgeführt. Die berechneten Werte werden anschließend per `MPI_Broadcast` zu den übrigen Prozessoren kopiert (siehe auch Abbildung 4.1).

Der berechnete Octree ist per `MPI_Broadcast` nur schwer zu übertragen. Die Übertragung von Derived Types mit MPI ist komplex und erfordert die Erstellung eines neuen MPI-Datentyps [21]. Dies ist prinzipiell möglich. Die im Octree verwendeten Zeiger sind jedoch nur im Speicher des ersten Prozessors gültig und verlieren bei der Kopie ihre Gültigkeit, sodass der Octree auf den übrigen Prozessoren unbrauchbar ist.

Das MPI-Modul von EIRENE ist daher erweitert worden. Nachdem alle Broadcasts abgeschlossen sind, wird der aus Abbildung 4.2 bekannte Ablauf zum Aufbau des Octrees in den übrigen Prozessen wiederholt. Dabei wird die Ausführung des `TIMEA0`-Codes ausgespart, da dessen Ergebnisse bereits kopiert wurden. Der Baum ist danach im Speicher jedes Prozesses vorhanden.

Da die Geometrie keine zeitliche Änderung erfährt und somit die Octrees während eines EIRENE-Laufs nicht ihre Gültigkeit verlieren, können die Bäume unsynchronisiert in den lokalen Speichern abgelegt werden.

4.3. Nutzung des Baumes

Nach Aufbau des Octrees in der Routine `TIMEA0` wird die Simulation gestartet. Die Routine `TIMEA1` soll während der Simulation die Schnittpunkte zwischen Teilchenflugbahnen und Geometrieflächen - Additional Surfaces - , genauer: die Flugzeit zu diesem Schnittpunkt, berechnen, sofern diese existieren. Alle weiteren zur Simulation notwendigen Schritte und Abläufe sind nicht Teil von `TIMEA1`. Der Programmablauf folgt der Erklärung in Abschnitt 4.1, S. 21, grafisch dargestellt in Abbildung 4.1, S. 22.

Die grundlegende Idee zur Verkürzung der Laufzeit ist eine Reduktion der Menge der zu testenden Flächen auf eine minimale Anzahl. Da die Teilchenflugbahn einen Startpunkt A aufweist, wird anhand dieses Punktes in einer lokalen, abgegrenzten Umgebung nach Schnittpunkten mit Geometrieflächen gesucht.

Die Umgebung ist dabei das Octetvolumen, in dem der Punkt selbst liegt. Wird in der Umgebung kein Schnittpunkt gefunden, muss die in Flugrichtung benachbarte Umgebung nach Schnittpunkten durchsucht werden, d. h. A wird entlang der Flugbahn in eine neue Umgebung verschoben. Diese Verschiebung wird so lange wiederholt, bis entweder keine benachbarte Umgebung mehr existiert oder ein geeigneter Schnittpunkt gefunden wurde.

Diese Verschiebung entspricht dem in Kapitel 3 vorgestellten „Raytraversal“. Die Menge der in jeder Umgebung zu testenden Flächen ergibt sich aus den assoziierten Flächen des Octets, das der Umgebung entspricht.

Die Tests der Flächen auf Schnittpunkte mit der Flugbahn und die Berechnung der Flugzeit zu einem Schnittpunkt wurde bislang in einer globalen Umgebung durch lineare Suche über alle Geometrieflächen in `TIMEA1` ausgeführt.

Da dieser Code mehrere Jahrzehnte alt ist, können potentiell darin enthaltene Seiteneffekte[2] nicht ausgeschlossen werden. Der mögliche Verlust durch eine Reimplementierung ist zu vermeiden. Es soll daher möglichst viel der ursprünglichen Struktur und des Quelltextes erhalten bleiben. Des weiteren soll die neue Funktionalität für den Nutzer optional ohne Quelltextänderung ein- bzw. ausschaltbar sein.

Abbildung 4.10 beschreibt grafisch den Ablauf der Berechnungen. Das Ablaufdiagramm zeigt den bereits veränderten Programmablauf nach der Abwandlung zur Nutzung des Octrees und soll im Weiteren näher erläutert werden.

Nicht alle möglichen „Additional Surfaces“ sind im Octree assoziiert; nur Drei-, Vier- und Fünfecke werden berücksichtigt ($3 \leq$`RLB`≤ 5, s. Abs. 4.2). Nicht assoziierte Flächen sind im Feld `notOCSurfs` gespeichert (vgl. Abs. 4.2) und werden bei einem Aufruf der Routine `TIMEA1` zuerst auf mögliche Schnittpunkte untersucht. Sollte der Octree nicht verwendet werden, enthält das Feld alle Additional Surfaces und sucht somit gemäß der alten Implementierung nach den Schnittpunkten.

Um für die Octree-Nutzung den Code, der bisher mit Hilfe der linearen Suche Schnittpunkte berechnet hat, wiederverwenden zu können, ist dieser in eine neue Routine `CheckInter` auszulagern. Die Routine `TIMEA1` ruft die Routine `CheckInter` mit denselben Parametern auf, die sie bei ihrem Aufruf erhalten hat (Aufruf s. Abb. 4.1, S. 22). Zusätzlich wird ein Feld mit den Indizes der zu testenden Flächen übergeben. Folgende Parameter werden durch das Monte-Carlo-Experiment generiert:

1. Startpunkt (Ortsvektor der Geraden)
2. Richtung des Fluges (Richtungsvektor der Geraden)
3. Geschwindigkeit des Teilchens

Die Abbildung 4.11 stellt den Ablauf der Routine `CheckInter` - bzw. den Ausschnitt von `TIMEA1` - vor und nach der Einführung des Octrees dar. Vor der Einführung wurde eine Schleife über alle Flächenindizes genutzt, um die zu untersuchende Fläche mit dem Index `J` festzulegen. Der vorhandene Code zur Berechnung des Schnittpunktes wird auch nach der Einführung identisch weiter genutzt. Der Unterschied liegt in der Nutzung einer Index-Tabelle, die der Routine `CheckInter` als Feld übergeben wird. Die Schleife iteriert über die Einträge in dieser Tabelle und setzt den Flächenindex `J` mit dem im Feld eingetragenen gleich. Im Falle der Flächen außerhalb des Octrees entspricht das Feld in `CheckInter` dem Feld `notOCSurfs`.

Falls in den Nicht-Octree-Flächen ein Schnittpunkt gefunden wird, wird dieser Kandidat zur weiteren Suche nach noch näher gelegenen Kandidaten zwischengespeichert. Wird kein Octree genutzt, wird der Schnittpunkt und die Flugzeit an die rufende Routine zurück gegeben und `TIMEA1` terminiert.

[2] Seiteneffekte: Prozeduren, die nur auf ihren Eingabedaten arbeiten und Daten der Umgebung unangetastet lassen, nennt man „seiteneffektfrei“ [22]. Beispiel: die Inkrementfunktion ist seiteneffektfrei, kann aber Seiteneffekte auslösen, wenn sie auf Zeigern arbeitet.

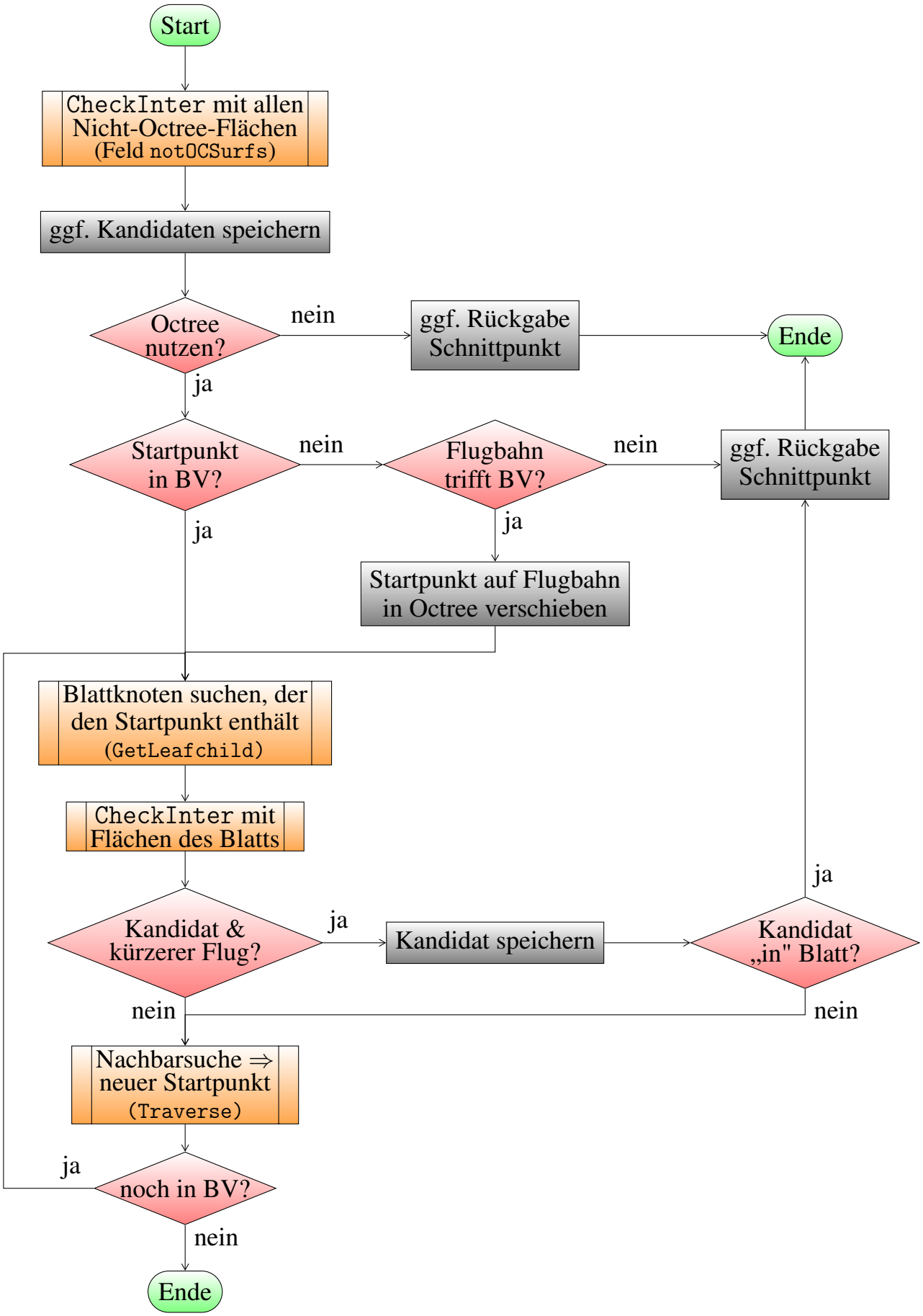
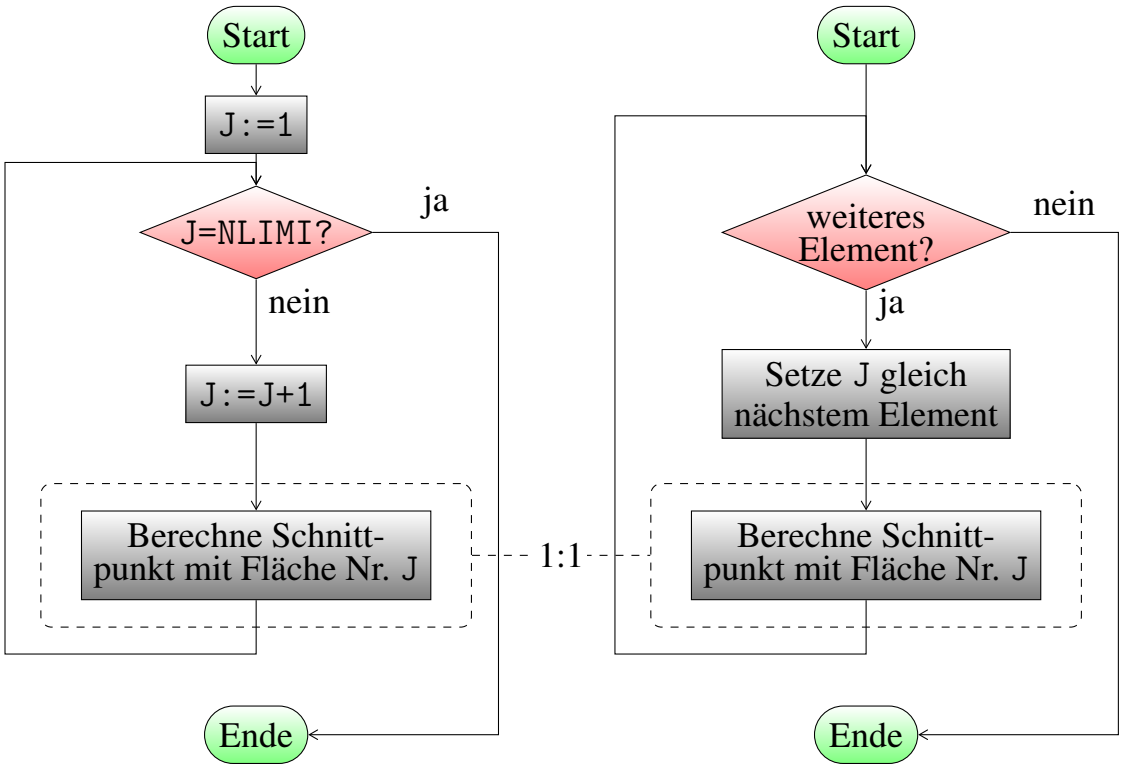


Abbildung 4.10.: Ablaufdiagramm TIMEA1 mit Octree.



(a) TIMEA1 (b) CheckInter

Abbildung 4.11.: Ablaufdiagramm der Schnittpunkt-Berechnungen (a) vor und (b) nach Einführung des Octrees. Speichern des Punktes mit der kürzesten Flugzeit in Berechnung integriert. NLIMI $\hat{=}$ Anzahl Add. Surf.

Alle weiteren Operationen nutzen eine Kopie A' des Startpunkts A der Flugbahn, der dem Aufpunkt der Geraden entspricht.

Bei Nutzung eines Octrees wird zunächst überprüft, ob A' innerhalb des Octree-Volumens liegt. Die Prüfung wird mittels der Routine CheckVolume des Moduls EIRMOD_OCTREE durchgeführt. Diese überprüft, ob komponentenweise $B_1 \leq A' \leq B_2$ gilt. Nur wenn dies der Fall ist, liegt der Aufpunkt innerhalb des Octree-Volumens. Für den gegenteiligen Fall muss geprüft werden, ob die Flugbahn des Teilchens das Volumen trifft. Dazu wird erneut die bereits in Abschnitt 4.2.4 genutzte Routine CheckBlock verwendet.

Wenn sie nicht trifft, wird ein ggf. zwischengespeicherter Kandidat zurück gegeben und TIMEA1 terminiert. Andernfalls wird der Startpunkt A' mit dem von CheckBlock berechneten Schnittpunkt von Flugbahn und Octree-Volumen gleichgesetzt. Im weiteren kann also davon ausgegangen werden, dass der Startpunkt A' innerhalb des Octree-Volumens liegt.

Zur Verdeutlichung der nun folgenden Schnittpunktsuche ist es notwendig, das im Abschnitt 4.2.3 bereits grundlegend eingeführte Konzept der Location Codes in ihrer Anwendung zu erläutern.

4.3.1. Punktsuche mit Location Codes

Um das kleinstmögliche Octet zu finden, das den Punkt A' enthält und dessen Flächen auf Schnittpunkte getestet werden sollen, wird eine sogenannte Punktsuche durchgeführt. Dazu wird die Methode `GetLeafchild` des Moduls `EIRMOD_OCTREE` genutzt. Zur Beschleunigung dieser Knotensuche im Baum werden Location Codes eingesetzt.

Die Routine erhält als Übergaben den Baum `tree` vom Typ `ocTree` und den Punkt, den das gesuchte Octet des Baumes enthalten soll. Die grundlegende Idee von [1] zur Suche nach dem Knoten besteht darin, aus dem Punkt durch eine Transferfunktion einen Location Code zu erhalten, der mit dem des gesuchten Knotens übereinstimmt. Gleichzeitig soll der Location Code erlauben, iterativ den Baum von der Wurzel aus fortschreitend zu dem Knoten zu durchsuchen, sodass die Nutzung des Stacks für jeden Rekursionsaufruf vollständig vermieden werden kann. [1] betont das die eingesetzten elementaren Operationen nach Möglichkeit modernen Prozessoren erlauben, innerhalb ihrer Caches zu arbeiten.

Gesucht ist eine Funktion zur Bestimmung eines Location Codes $LC = f(A')$. Nach [1] bestimmt sich diese Funktion als

$$LC_i = \left(\left\lfloor maxvalue * \frac{A'_i - \tilde{B_{1i}}}{length_i} \right\rfloor \right)_2, \; i = x,y,z$$

Durch $c = \frac{A'_i - \tilde{B_{1i}}}{length_i}$ wird jede Komponente von A' auf ein Intervall $[0,1]$ abgebildet, das der Normierung des Wertebereiches des Octrees auf $[0,1] \times [0,1] \times [0,1]$ entspricht. `length` ist der Vektor der Kantenlängen des Boundary Volumes, der sich durch $length = \tilde{B_2} - \tilde{B_1}$ bestimmt. Die Bestimmung von $\tilde{B_1}$ und $\tilde{B_2}$ ist bereits in Abschnitt 4.2.1 erörtert worden.

Die Aufweitung des Boundary Volumes ist bedingt durch die Fließkomma-Arithmetik, die bei der Berechnung von c zu beachten ist. Alle Punkte A' werden durch die Aufweitung des Volumens um $2 * 10^{-12}$ innerhalb der Funktion zur Berechnung des Location Codes scheinbar nach innen verschoben. Eventuelle Rechenungenauigkeiten doppelt genauer Fließkommazahlen im Rahmen von 10^{-15} werden ausgeglichen, sodass auch Punkte sehr nahe am Rand des Octree-Volumens noch in korrekte Location Codes übersetzt werden.

Der Location Code ergibt sich als Produkt von c mit dem Wert der Variablen `maxvalue`, die wie `length` in `tree` gespeichert ist. Der Wert der Variablen berechnet sich aus der Anzahl der Ebenen im Baum: $maxvalue = 2^{layers-1}$; für einen Baum mit drei Ebenen wie in Abbildung 4.6, S. 31 beträgt demnach $maxvalue = 2^{3-1} = 4$.

Somit ist $d = maxvalue * \frac{A'_i - \tilde{B_{1i}}}{length_i}$ eine Fließkommazahl zwischen $d \in [0, maxvalue]$. Die Umwandlung in einen Ganzzahlwert erfolgt in Fortran mittels der Funktion `int()`, hier geschrieben als untere Gaußklammer.

Die Darstellung der Ganzzahl als Binärfolge ergibt den gesuchten Location Code, der `layers`-viele Stellen hat. Es ergeben sich durch die Umwandlung der Fließkommazahl $d \in [0, maxvalue]$ in Ganzzahlen insgesamt $2^{layers} + 1$ „Grenzen“, die 2^{layers} Teilabschnitte einer Kante des Octetvolumens beschreiben.

Aus diesem Grund funktionieren Location Codes nur für Bäume mit gleichgroßen Unterteilungen - z. B. am Raummittelpunkt -, da die Ganzzahlen einen äquidistanten Abstand zueinander haben und somit keine unregelmäßigen, nicht äquidistanten Abstände abbilden können. Eine Unterteilung am Objektmittelpunkt und die Verwendung von Location Codes, wie sie hier beschrieben wurden, schließen sich daher gegenseitig aus.

Der berechnete Location Code $LC = (LC_x, LC_y, LC_z)^T$ lässt sich zur iterativen Suche des Knotens verwenden. Wie bereits in Abschnitt 4.2.3 beschrieben, entspricht jede Binärstelle einer Ebene des Baumes. Wird auf einer Ebene e des Baumes der nächste Kindknoten gesucht, in dem der Punkt enthalten ist, berechnet die Abbildungsvorschrift 4.3 den Index i des Kindes:

$$i = 4 * (LC_z(e))_{10} + 2 * (LC_y(e))_{10} + (LC_x(e))_{10} + 1 \tag{4.3}$$

$LC_{x,y,z}(e)$ bezeichnet dabei die Binärziffer an der Stelle von 2^e des Location Codes. Die Suche in `GetLeafchild` nach dem kleinsten Blattknoten, der den Punkt A' enthält erfolgt iterativ. Ausgehend von der Wurzel des Baumes wird mit Gleichung 4.3 der Index des Kindes bestimmt, in dem der Punkt liegt. Hat dieses Kind wieder Kinder, erfolgt erneut eine Indexberechnung, bis ein Blattknoten erreicht wird. Dieser muss nicht notwendigerweise auf der untersten Ebene liegen, wenn ein auf höherer Ebene gelegener Knoten nicht unterteilt wurde. Der Knoten ist dann das gesuchte kleinste Octet.

4.3.2. Durchlaufen des Baumes

Ist der Knoten für den Startpunkt A' gefunden, wird das im Knoten gespeicherte Feld `surfaces` zusammen mit den Parametern, die `TIMEA1` übergeben wurden, an die Routine `CheckInter` übergeben. Die Routine sucht in den Flächen, mit deren übergebenen Indizes, nach einem Schnittpunkt. Dabei nutzt sie als Aufpunkt der Flugbahn nicht den Punkt A', sondern den ursprünglich der Routine `TIMEA1` übergebenen Punkt A.

Wird ein Schnittpunkt gefunden und ist die Flugzeit dorthin kürzer als zum ggf. bisherigen Kandidaten und liegt dieser Punkt innerhalb des Volumens des Octets, in dem auch der Startpunkt liegt, kann die Suche beendet werden. Es kann keinen näher gelegenen Schnittpunkt mehr geben. Es folgt Rückgabe und Terminierung von `TIMEA1`.

Da eine Fläche ein Octet auch nur durchstoßen kann, können Schnittpunkte gefunden werden, die nicht innerhalb dieses Octets liegen. Da immer die Fläche als Ganzes getestet wird, muss dieser Fall durch eine Prüfung, ob der gefundene Schnittpunkt im „richtigen“ Octet liegt, behandelt werden. Wenn er im Octet liegt, wurden alle umgebenden Flächen getestet und von `CheckInter` bereits verworfen. Alle in vorherigen Octets getesteten Flächen wurden ebenfalls bereits verworfen. Jedes Octet, das nach dem aktuellen getestet wird, muss eine längere Flugzeit zur Folge haben. Es ist daher unter der Voraussetzung, dass der gefundene Schnittpunkt innerhalb des Octets mit den untersuchten Flächen liegt, nicht möglich, eine noch kürzere Flugzeit zu einem anderen Schnittpunkt zu finden.

Sofern kein Kandidat gefunden wurde oder der Kandidat nicht im Octet liegt, folgt der Traversal-Schritt. Dieser bestimmt, wo als nächstes nach Schnittpunkten gesucht wird.

4.3.3. Nachbarsuche, Traversal-Schritt

Die Fachliteratur zum Einsatz von Octrees im Raytracing kennt viele verschiedene Arten, den Ray Traversal durchzuführen [5, 1, 15, 23, 24, 25]. Zwei Ansätze bilden dabei die Grundlage der verschiedenen Verfahren:

1. Top-Down-Ansatz: Suche des nächsten Knotens von der Wurzel aus und
2. Bottom-Up-Ansatz: Suche des nächsten Knotens vom aktuellen Knoten aus.

Mit Location Codes ist es möglich, einen effizienten Bottom-Up-Ansatz zu implementieren [1]. Die Implementierung für EIRENE nutzt jedoch einen Top-Down-Ansatz. Aufgrund der auf einstellige Werte beschränkten Höhe h des Baumes (s. Abs. 4.2.1) ist die bei hohen Bäumen ($h \gg 5$) problematische dauerhafte Suche von der Wurzel aus nutzbar. Die Kombination der Ansätze von [26, 1] liefert das im Folgenden näher beschriebene Verfahren.

Gesucht wird ein Punkt C, der auf der Flugbahn des Teilchens liegt und im Nachbaroctet des Octets liegt, in dem der Punkt A' enthalten ist. Sofern dieser Punkt C außerhalb des Boundary Volumes des Octrees liegt, ist die Schnittpunktsuche ergebnislos abzubrechen. Andernfalls ist zu diesem Punkt der Knoten zu bestimmen, der das kleinste, den Punkt C einschließende Octet beinhaltet.

Diese Aufgabe ist bereits durch die Punktsuche mittels Location Codes gelöst. Bevor ein neuer Suchlauf gestartet werden kann, muss also der Punkt C bestimmt werden. Alle folgenden Beschreibungen werden durch die Routine `Traverse` des Moduls `EIRMOD_OCTREE` gekapselt.

Die grundlegende Idee von [26] ist, dass, ausgehend von einem Schnittpunkt S der Flugbahn und einer Seitenfläche des aktuellen Octets, der gesuchte Punkt C ein kurzes Stück weiter entlang der Flugbahn liegt.

In der Implementierung für EIRENE wird das „kurze Stück“ über den Parameter `shortest` und `factor` des Baumes definiert. `shortest` ist die Länge der kürzesten Kante aller Kanten von Octetvolumen. `factor` ist der Divisor der Variablen `shortest`, somit wird `shortest` nicht als Länge sondern als der Wert des Quotienten von Länge und Faktor gespeichert. Die Bestimmung von `shortest` erfolgt beim Bau des Octree, siehe Abschnitt 4.2.2.

Ein typischer Wert für `factor` ist 10^3, sodass keine Gefahr besteht, dass durch ein zu langes „kurzes Stück“ ein Octet übersprungen werden kann. Ausgeschlossen ist so ebenfalls, dass `factor` zu klein ist und `shortest` zu nahe der Rechengenauigkeit von 10^{-15} liegt, da eine kürzeste Kantenlänge von 10^{-12} unwahrscheinlich ist. Der Punkt C wird weit genug in das neue Octet verschoben, sodass die Punktsuche ohne die Gefahr von Rechenungenauigkeiten funktioniert.

Die Berechnung der Schnittpunkte mit den Ebenen der Seitenflächen erfolgt nach dem in Abschnitt 4.2.4 erläuterten Verfahren zum Durchstoßtest von Flächenkanten mit Seitenflächen, nur dass die Normalenvektoren nach Außen zeigen, da aus dem Inneren nach Außen „geflogen“ wird. Der Schnittpunkt S_i innerhalb des Octetvolumens mit der kürzesten Entfernung λ_i zu A' bildet die Basis zur Berechnung von C.

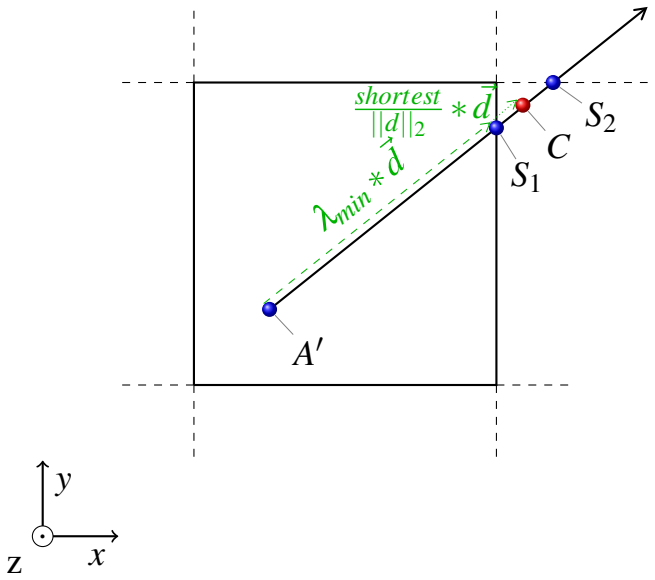


Abbildung 4.12.: Nachbarsuche, Ansicht von oben (Projektion auf XY-Ebene)
S_3 nicht eingezeichnet. $\vec{d}$ ist Richtungsvektor der Geraden der Flugbahn

Durch die Kenntnis des Parameters $\lambda_i = \lambda_{min}$ folgt die Vektoraddition $C = A' + \lambda_{min} * d + \frac{shortest}{||d||_2} * d$, dabei ist d der Richtungsvektor der Flugbahn (siehe Abbildung 4.12).
Die Punktsuche nach C als neuem A' erfolgt im nächsten Schritt.

Im nächsten Kapitel ist die beschriebene Implementierung auf Funktion und Korrektheit zu prüfen sowie eine Laufzeitanalyse durchzuführen, ob die Nutzung eines Octrees in unterschiedlichen Fällen eine Zeitersparnis bedeutet.

5. Test und Analyse

Die Ergebnisse der Simulation in EIRENE müssen bei aktivierter Octree-Erweiterung mit denen der linearen Suche der ursprünglichen Implementierung exakt übereinstimmen. Die vom Betriebssystem generierten Pseudo-Zufallszahlen, welche EIRENE zur Monte-Carlo-Simulation nutzt, können durch den entsprechenden „Seed-Startwert“ für den Generator entweder gleich oder verschieden ausfallen. Bei gleichem Startwert ergeben sich die exakt gleichen Folgen von Zufallszahlen. Welche Variante EIRENE nutzt, ist durch einen logischen Schalter in der Eingabedatei steuerbar.

Für die Programmverifikation und die Vergleichbarkeit der Ergebnisse beider Implementierungen sind Läufe notwendig, die dieselben Folgen von Zufallszahlen nutzen. Da die Monte-Carlo-Experimente in beiden Fällen konform ablaufen, müssen berechnete Schnittpunkte und Ergebnisse genau übereinstimmen.

Für eine Beurteilung der Laufzeiteinsparungen durch die Aktivierung des Octree ist diese Methode jedoch ungeeignet. Wird immer mit denselben Folgen von Zufallszahlen gerechnet, ergibt sich für jeden Test eine ähnliche Laufzeitbewertung, die nur von der Auslastung des Computers abhängt. Zur praktischen Beurteilung der Verbesserung, und um möglichst unterschiedliche Serien von Testläufen zu erhalten, sind daher für jeden neuen Test unkorrelierte Folgen von Zufallszahlen zu wählen. Die Ergebnisse und Schnittpunkte sind dann jedoch nicht mehr exakt vergleichbar, sondern nur nach statistisch zu analysieren.

Die Simulation der Plasmen wird in der realen Anwendung in der Regel nicht mit weniger als 5.000 Teilchentrajektorien durchgeführt. Oft sind 100.000 und mehr Teilchen für eine aussagekräftige (= statistisch signifikante) Simulation notwendig, da die physikalischen Größen mit einer zu geringen Anzahl and Stichproben als statistisch unsicher zu betrachten sind. Eine quantitative Aussage hierzu liefern die „empirischen Varianzen”, die von EIRENE für jede berechnete Größe aus der Stichprobe von Trajektorien direkt ebenfalls mit ausgewertet werden. Die Laufzeittests sind daher mit verschiedenen Anzahlen von Teilchenbahnen in vergleichbarer Höhe durchgeführt worden.

Zur Verifikation und Analyse der Implementierung werden verschiedene Geometrien genutzt (vgl. Tab. 5.1). Dazu zählen vier Zylindermodelle CYLINDER.P12 (Abb. D.1), CYLINDER.P24 (Abb. D.2), CYLINDER.P48 (Abb. D.3) und CYLINDER.EIR(s. Abb. D.4) sowie drei unterschiedliche Modelle eines ITER-Portplugs M1 (Abb. D.5), M1BAF (Abb. D.6) und M1M4 (Abb. D.7).

Fallname	Dreiecke	Besonderheiten
CYLINDER.P12	1086	
CYLINDER.P12.PLASMA	1086	Plasma im Inneren
CYLINDER.P24	4630	
CYLINDER.P24.PLASMA	4630	Plasma im Inneren
CYLINDER.P48	18212	
CYLINDER.EIR	0	Flächen 2. Ord.
M1	4325	reale Geometrie
M1BAF	5967	reale Geometrie
M1M4	7161	reale Geometrie

Tabelle 5.1.: Übersicht der Testgeometrien zur Analyse des Laufzeitverhaltens von Linearer Suche und Octree-Erweiterung

Bis auf CYLINDER.EIR sind alle Geometrien unterschiedlich große Dreiecksmodelle (vgl. Tab. 5.1). CYLINDER.EIR modelliert den Zylinder mit Hilfe einer Fläche 2. Ordnung sowie zwei Ebenen als Stirnseiten. Der Fall liefert einen Anhaltspunkt zur Beurteilung, ob die Implementierung mittels Octree konkurrenzfähig ist im Vergleich mit dieser Beschreibung derselben Geometrie und der gleichen Anzahl von Teilchenverfolgungen.

CYLINDER.P12,24,48 definieren den Zylinder als Dreiecksgitter mit unterschiedlicher Auflösung: Kreisförmige, regelmäßige Schnitte des Zylinders als Fläche zweiter Ordnung werden mit einer Anzahl x Sekanten als Polygone approximiert. Jeder approximierte Ring ist um eine halbe Sekante gegenüber seinem Nachbarn um die Mittelachse des Zylinders rotiert. Die Dreiecke des Zylindermodells ergeben sich durch Verbinden einer Sekante auf einem Ring mit dem auf dem Nachbarring gegenüberliegenden Schnittpunkt zweier Sekanten. Die Anzahl der Sekanten x ist in den Modellnamen angegeben.

Die Triangulationen der Geometrien M1, M1BAF und M1M4 wurden mit ANSYS erstellt (s. Kap. 2).

Die Zylindergeometrien sind als Modelle mit annähernd gleichmäßig verteilten und gleich großen Dreiecksflächen geeignet zur Messung der Güte des Codes in Bezug auf das Laufzeitverhalten für die realistischen Geometrien der Port-Plugs, die nicht gleichmäßig verteilt sind und unterschiedlich große Flächen aufweisen.

Zu erwarten ist eine hohe bis sehr hohe Laufzeitreduktion für gleichmäßige Geometrien; Für ungleichmäßig verteilte Geometrien ist der zu erwartende Erfolg vor der Durchführung der Tests unbekannt.

Die Testfälle CYLINDER.P12.PLASMA und CYLINDER.P24.PLASMA entsprechen den Geometrien CYLINDER.P12 bzw. CYLINDER.P24. Im Gegensatz zu diesen simulieren sie das Experiment mit einem Hintergrundmedium, quasi „im Plasmabetrieb“ anstatt mit Vakuum im Inneren der Geometrie. Durch die Stoßprozesse im Inneren verringert sich die Wahrscheinlichkeit für ein Teilchen eine Geometriefläche zu treffen.

Dies läßt erwarten, dass bei gleicher Anzahl Teilchenverfolgungen weniger Laufzeit benötigt wird.

Im Folgenden wird zunächst die Korrektheit der Octree-Erweiterung durch Vergleich von Simulations- und Zwischenergebnissen dargelegt. Anhand statistischer Messungen wird anschließend die Implementierung des Octrees auf ihr Laufzeitverhalten im Vergleich zur linearen Suche untersucht. Anhang A beschreibt die Software EGView3D, die zur Programmverifikation im Rahmen dieser vorliegenden Arbeit erweitert wurde.

5.1. Programmverifikation

Die Verifikation erfolgt anhand der Ausgabe der Simulationsergebnisse und die Zwischenergebnisse in Form der Schnittpunkte der verfolgten Teilchen. Dafür wurden die Dreiecksgeometrien CYLINDER.P12 (Abb. D.1) und CYLINDER.P24 (Abb. D.2) herangezogen. Weitere Geometrien sind nicht für die Programmverifikation genutzt worden.

Die Geometrie CYLINDER.P12 besteht aus 1.086 Dreiecken, CYLINDER.P24 aus 4630. Es wurden Testläufe mit gleichen Zufallszahlen, mit und ohne Octree sowie jeweils 10, 100, 500, 1.000, 5.000, 10.000, 50.000 und 100.000 Teilchen gestartet. Die Ergebnisse der Läufe mit und ohne Octree stimmen jeweils exakt überein, sodass davon ausgegangen werden kann, dass die Octree-Implementierung korrekte Ergebnisse erzeugt.

Auch die Nutzung der Optimierung des Codes durch die Compiler ergibt keine Unterschiede sowohl bei aktivem wie inaktivem Octree als auch im Vergleich der Ergebnisse mit den Programmen ohne Optimierung durch den Compiler.

Das Programm arbeitet auch unter Verwendung von MPI zum verteilten Rechnen auf mehreren Prozessoren korrekt. überprüft wurde dies durch Testläufe mit der Geometrie CYLINDER.P12 mit aktivem Octree, 10.000 Teilchen und aktivem logischen Schalter `NLIDENT`, der bewirkt, dass auf allen Prozessoren die selben Folgen von Zufallszahlen genutzt werden. Die Ergebnisse der Prozessoren stimmen exakt überein.

5.2. Analyse des Laufzeitverhaltens

Zur Beurteilung der Güte der Optimierung müssen verschiedene Aspekte betrachtet werden:

1. die gemessene Zeitersparnis ist ein wichtiger Indikator,
2. der notwendige zusätzliche Speicherbedarf muss dabei in Betracht gezogen werden.

Wichtig ist, die in Abschnitt 4.2.1 angesprochene Problematik bzgl. der Flächenzahl in den Blattknoten an realen Beispielgeometrien zu analysieren.

Um vergleichbare Ergebnisse zu erhalten, sind zwei nahezu identische Maschinen genutzt worden, deren Konfiguration in Tabelle C.1 im Anhang auflistet ist. Der wesentliche Unterschied bestand in den eingesetzten Fortran-Compilern, a) GNU gfortran und b) Intel ifort. Auf beiden Maschinen wurden die Testläufe mit und ohne Optimierungen durch den Compiler ausgeführt.

Fallname	gfortran			ifort		
	mittl. Laufz. in [s]		Reduktion	mittl. Laufz. in [s]		Reduktion
	Lin. Such.	Octree		Lin. Such.	Octree	
CYLINDER.P12	782	114	85,4%	797	81	89,8%
CYL.P12.PLASMA	377	61	83,8%	362	45	87,7%
CYLINDER.P24	6827	393	94,2%	7093	248	96,5%
CYL.P24.PLASMA	3607	195	94,6%	4075	138	96,6%
CYLINDER.P48	>7210	846	>88,2%	>7246	571	>92,1%
CYLINDER.EIR	33	33	-	14	14	-
M1	266	85	68,0%	288	122	57,5%
M1BAF	415	162	61,0%	431	230	46,6%
M1M4	549	268	51,2%	673	454	32,5%

Tabelle 5.2.: Laufzeitanalyse für 100.000 Teilchentrajektorien für alle Testfälle, gemittelt über 10 Läufe. Rote Zahlen für die beste Laufzeit.
CYLINDER.EIR ohne Einsparung da keine Dreiecke. Mit Compiler-Optimierung simuliert. Einsparung für CYLINDER.P48 ist real größer, kann aufgrund der Laufzeitüberschreitung nicht genauer angegeben werden (vgl. Tabelle C.2).

Zur Erhebung der Messdaten wurden für die neun Testfälle aus Tabelle 5.1 jeweils zehn Simulationen pro Teilchenanzahl mit aktivem oder inaktivem Octree gestartet. Die Anzahl der Teilchenverfolgungen richtete sich nach den unter realen Anwendungen genutzten Anzahlen von mindestens 5.000 Teilchen. Es wurde daher mit 5.000, 10.000, 50.000 und 100.000 Teilchen gerechnet. Bei allen Simulationen ist die Laufzeit von EIRENE in den Eingabedaten auf 7.200 Sekunden Laufzeit begrenzt. Die Fälle CYLINDER.P24 und CYLINDER.P48 ohne aktiven Octree wurden für einige Parameterkombinationen durch die Begrenzung der Laufzeit auf 7.200 Sekunden vorzeitig durch den EIRENE-Code beendet.

Die vollständige Übersicht der gemittelten Werte ist im Anhang für den Compiler GNU gfortran in Tabelle C.2 und für den Compiler Intel ifort in Tabelle C.3 gelistet. Die auf dieser Datenbasis erstellten Graphen C.1 bis C.9 zeigen die gemittelte benötigte Laufzeit in Sekunden.

In Tabelle 5.2 sind die gemittelten Laufzeiten für 100.000 Teilchentrajektorien für beide Compiler mit aktiven Compiler-Optimierungen aufgeführt. Diese Anzahl an Bahnen entspricht einer Standardanzahl für Simulationen in der Praxis.

Die Laufzeitstatistik zeigt, dass für die gleichmäßig verteilten Dreiecksflächen der Zylindergeometrien das ausführbare Programm des ifort-Compilers besser geeignet ist. Der gfortran-Compiler erzeugt für die realen Geometrien M1, M1BAF und M1M4 den effizienteren Code. Zwar ist dieser Zusammenhang auch in den übrigen Messdaten in den Tabellen C.2 und C.3 erkennbar, jedoch sind aufgrund der relativ kleinen Stichprobengröße von zehn Läufen pro Kombination diese Aussagen statistisch nicht hinreichend gesichert und erfordern weitere Analysen. Aufgrund der teilweise extremen Laufzeiten ohne Octree (>7.200s) wurde die Stichprobengröße beibehalten.

Bei beiden Compilern ist die Tendenz der mit einem Faktor 6 bis mindestens 12 (80-96%) sehr gut beschleunigten Simulation der Zylindergeometrien zu erkennen. Die mit einem Faktor ca. 2 bis 3 (50-66%) deutlich geringere Beschleunigung für die realen Testfälle der Port Plugs M1, M1BAF und M1M4 ist ebenso für beide Compiler zutreffend. Diese Tendenzen spiegeln sich ebenfalls in den übrigen Laufzeittests mit anderen Parameter-Kombinationen wieder (vgl. Tab. C.2 & C.3) Die Zahl der Dreiecke kann keinen Einfluss auf die Laufzeit-Ersparnis nehmen, denn CYLINDER.P48 und CYLINDER.P24 zeigen in ähnlich großen bzw. noch größeren Geometrien deutlich höhere Einsparungen.

Der Vergleich zwischen den einzelnen Zylinder-Geometrien zeigt, dass die Modellierung mit Flächen 2. Ordnung im Fall CYLINDER.EIR effektiver ist als die Verwendung von Dreiecken. Für die Port-Plug-Geometrien ist eine solche Vereinfachung nicht oder nur sehr mühsam zu erreichen, sodass die Beschleunigung durch die Octree-Erweiterung um mindestens einen Faktor 2 bereits ein signifikanter Fortschritt gegenüber der linearen Suche in den detailreichen Dreiecksgeometrien ist. Alle Ergebnisse der Monte Carlo Simulation sind dann, bei gleicher Rechenzeit, bereits um mindestens einen Faktor $\sqrt{2}$ genauer.

Über die genaue Ursache der Diskrepanz der Beschleunigungsfaktoren zwischen Zylindergeometrien und Port-Plug-Geometrien kann hier keine gesicherte Angabe gemacht werden. In Abbildung 5.1 ist für die realen Testgeometrien eine signifikante Anzahl von Blattknoten der untersten Ebene vorhanden, die mehr als elf Flächen beinhalten.

Ist das fünf- bis sechsfache dieser Menge pro Blattknoten unproblematisch, wie der Fall CYLINDER.P24 in der Abbildung 5.1 zeigt, so sind zehn- bis fünfzigfache Mengen eine mögliche Ursache für die Leistungseinbußen, die in Tabelle C.2 und C.3 dokumentiert sind.

Zur Abhilfe sind zwei Möglichkeiten in der weiteren Entwicklung zu untersuchen:

1. keine direkte oder nur schwache Kopplung der Baumhöhe h an die Flächenzahl, damit maximal 10 Flächen (obere Schranke a) pro Knoten assoziiert werden können oder
2. Teilung von Octets am „Objekt-Mittelpunkt“ zur annähernden Gleichverteilung der Flächen.

Während die erste Variante die Weiternutzung der Location-Codes erlaubt, kann im zweiten Fall dieses Konstrukt nicht mehr in der bisherigen Form angewendet werden, da die Location Codes darauf beruhen, dass ein Volumen gleichmäßig geteilt wird. Die Binärcodierung der Zugehörigkeit zu einer Hälfte in der jeweiligen Raumrichtung ist bei unregelmäßig aufgeteilten Octetvolumen nicht mehr möglich. Zur Nutzung von Location Codes in der ersten Variante muss ausserdem die Binärzahl deutlich mehr Stellen aufweisen, da keine vorab bestimmte Höhe h des Baumes die Ziffernzahl begrenzt. Einer Imbalance des Baumes würde nicht mehr entgegen gewirkt.

Ein Ansatz zur Realisierung ist die Begrenzung der Höhe auf $h = 32$. Diese Grenze ist aufgrund der Größe von 32 Bit einer Ganzzahl, die zur Speicherung der Binärzahl genutzt wird, gewählt. Da das Kriterium, ob ein Blattknoten erreicht worden ist, darauf beruht, ob der untersuchte Knoten Kinder hat oder nicht, ist die Länge der Binärzahl nicht fest vorgeschrieben.

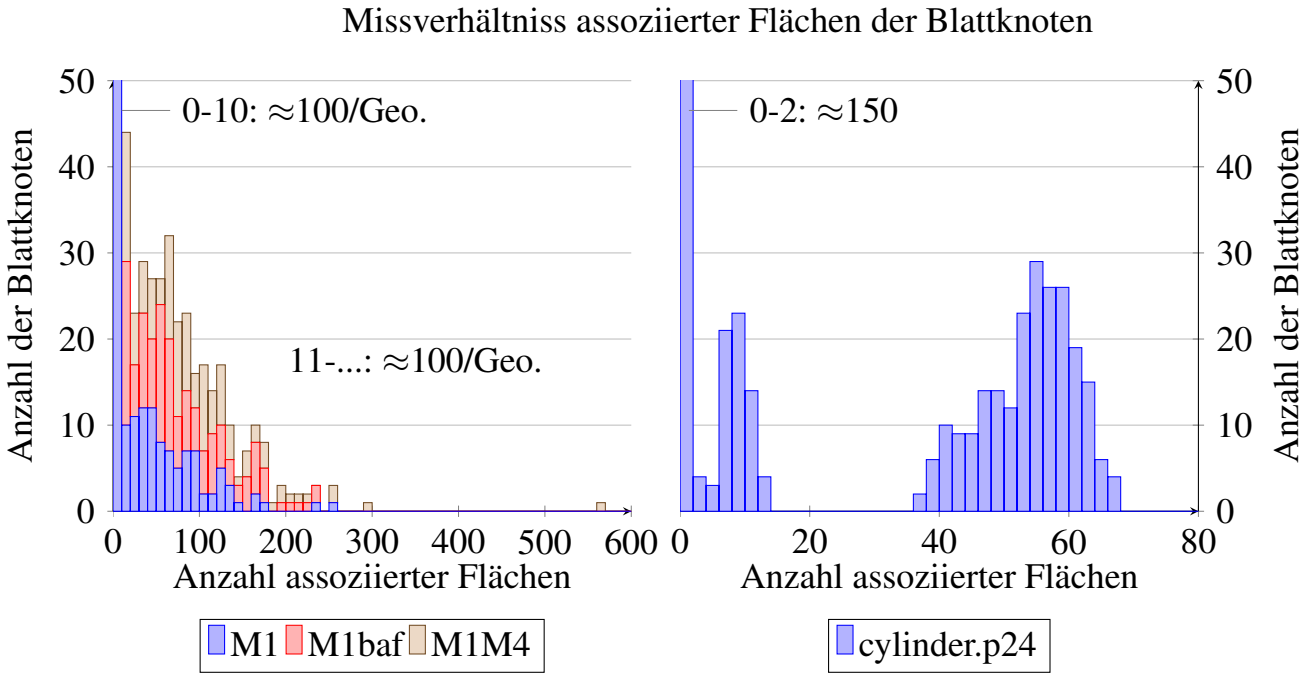


Abbildung 5.1.: Missverhältnis assoziierter Flächen zu Blattknoten.
In den Testfällen links ist jeder 4te Blattknoten „überfüllt“ mit 100-500 Flächen, rechts jeder 2te Blattknoten mit im Mittel 50 Flächen.
Links: Blattknoten der untersten Ebene pro Geometrie: ≈200 Stück.
Blattknoten >11 Flächen: ≈100, >60: ≈80, >100: ≈50 (!)
Rechts: Blattknoten der untersten Ebene: 448 Stück.
Blattknoten <20 Flächen: 225, >20 & <80: 223

Wichtig ist die weiterhin absteigende Nummerierung der Ebenen, um die Binärzahl von vorne nach hinten durchlaufen zu können (siehe Abschnitt 4.3.1). Zur Nachbarsuche müsste aufgrund der möglicherweise großen Höhe des Baumes auf den Bottom-Up-Ansatz gewechselt werden, wie in Abschnitt 4.3.3 erwähnt.

Die zweite Variante mit Teilung des Octets am „Objekt-Mittelpunkt“ lässt keine solche starke Imbalance zu, da die Flächen möglichst gleichmäßig aufgeteilt werden. Der Top-Down-Ansatz zur Nachbarschaftssuche ist in diesem Fall jedoch ineffizient, weshalb die Implementierung einer Bottom-Up-Suche zwingend erforderlich wird. Die Nichtanwendbarkeit der Location Codes würde rekursive Suchen notwendig machen, die bisher erfolgreich vermieden werden konnten.

Beide Varianten haben demnach sowohl Vor- wie Nachteile, die zukünftigen Anforderungen an EIRENE werden die Weiterentwicklung bestimmen.

Die beiden Fälle CYLINDER.P12.PLASMA und CYLINDER.P24.PLASMA sind als Geometrien mit denen ohne Plasma im Namen identisch, wurden jedoch während der Läufe nicht mit einem Vakuum, sondern mit einem Plasma im Inneren simuliert.

Die Erwartung ist, dass durch die zusätzlichen Prozesse im Plasma die Teilchen noch häufigeren Richtungsänderungen unterworfen sind und die Code-Optimierung hier eine noch stärkere Laufzeitreduktion bewirkt. Dies ist mit und ohne Plasma der Fall, wie aus Tabelle C.2 und C.3 im Anhang ersichtlich ist. Weshalb CYLINDER.P12.PLASMA eine deutlich geringere prozentuale Laufzeitersparnis hat als CYLINDER.P24.PLASMA, konnte bislang nicht geklärt werden.

Analyse der Speichernutzung

Die Nutzung eines Octrees für die Beschleunigung erfordert die Speicherung von Flächenindizes, Zeigern, etc. Eine Analyse des Mehrverbrauchs an Arbeitsspeicher zeigt jedoch, dass selbst für eine große Geometrie wie CYLINDER.P48 nur wenig Speicher zusätzlich verbraucht wird. Wird ohne Octree bei 8096 MiB RAM für einen Lauf der Geometrie CYLINDER.P48 32,3% = 2615 MiB des Speicherplatzes belegt, sind es mit Octree +0,3% = +24 MiB = 2639 MiB.

Obwohl der zusätzlich benötigte Speicherplatz für die Nutzung des Octrees unkritisch ist, fiel auf, daß der Gesamtspeicherbedarf mit 2600 MiB für eine Geometrie ohne Volumen-Diskretisierung des Rechengebietes erstaunlich hoch war. Weitere Analysen identifizierten zwei Integermatrizen der Größenordnung Flächenanzahl x Flächenanzahl als Ursache. Eine dieser Matrizen wird verwendet, um bei jeder Fläche explizit angeben zu können, welche anderen Flächen ein Teilchen sehen kann, wenn es von dieser Fläche startet. Die zweite Matrix legt für jede Fläche fest, ob bei der Schnittpunktberechnung einer Teilchenflugbahn mit einer sichtbaren Fläche zweiter Ordnung beide möglichen Schnittpunkte verwenden werden sollen. Beide Matrizen dienen einer einfachen Form der Optimierung. Es ist zu überlegen, ob diese Art der Optimierung beibehalten werden soll, oder ob man sie anders realisieren kann.

5.3. Code-Profiling

Zur Identifikation der Problemfelder in der Simulation der realen Geometrien wurden mit Hilfe des Profilers GNU gprof genauere Laufzeitstatistiken gesammelt. Diese geben einen Aufschluss über die Verteilung der Gesamtrechenzeit eines Laufs auf die einzelnen Code-Bereiche, wenn gewünscht mit exakter Angabe der Codezeile. Die Portplug-Geometrie M1 wurde als Analysebeispiel herangezogen und mit drei Teilchenquellen mit jeweils 100.000 Trajektorien simuliert.

Ohne Octree ergibt sich für die lineare Suche die unten stehende Verteilung der Rechenzeit. Die Schnittpunkt-Berechnung nimmt mehr als 75% der Laufzeit in Anspruch.

Profiling mit linearer Suche:

```
%     cumulative self    self    total
time  seconds    seconds calls   Ks/call Ks/call name
76.15 1792.63    1792.63 3801838 0.00    0.00    eirene_timea_checkinter_
22.11 2313.22    520.59  268762  0.00    0.00    eirene_stats0_
 0.29 2320.00    6.78    3801838 0.00    0.00    eirene_escape_
 0.24 2325.65    5.65    348457  0.00    0.00    eirene_folneut_
...
 0.00 2354.15    0.00    1       0.00    0.00    mpi_init_
```

Wird hingegen ein Octree genutzt, sinkt der relative Anteil der Rechenzeit für die Schnittpunkt-Berechnungen von ca. 75% auf ca. 30% ab (siehe unten). Die relative Laufzeitersparnis beträgt 86%. Allerdings fällt in beiden Profilings auf, daß ein weiteres Modul des EIRENE Codes einen deutlichen Anteil der Rechenzeit benötigt. Es handelt sich dabei um ein Modul, das für die Ergebnisgrößen die zugehörige Standardabweichung berechnet. Bei den Zylindergeometrien war die Berechnung der Standardabweichungen abgeschaltet. Aber in den Rechnungen mit den realistischeren Geometrien wurden die Standardabweichungen berechnet, um die Güte der Ergebnisse beurteilen zu können. Diese Berechnungen nehmen bei der linearen Suche einen Anteil von 22% der Rechenzeit ein. In gleicher Höhe findet sich diese Laufzeit auch im Profil mit aktivem Octree, jedoch wegen der erfolgreichen Optimierung des `TIMEA` Moduls mit einem relativ höheren Anteil von nun 64% der Gesamtlaufzeit.

Profiling mit Octree:

```
%     cumulative self    self     total
time  seconds    seconds calls    s/call s/call name
64.22 491.14     491.14  300005   0.00   0.00   eirene_stats0_
30.99 728.16     237.02  20276642 0.00   0.00   eirene_timea_checkinter_
 0.55 732.34     4.18    5010225  0.00   0.00   eirene_escape_
...
 0.00 764.80     0.00    1        0.00   0.00   mpi_init_
```

Zur weiteren Verbesserung der Laufzeit ist somit die Berechnung der Standardabweichung für flächengemittelte Ergebnisgrößen (wie Teilchen- und Energieflüsse, Sputter-Raten, etc.) im EIRENE-Code zu optimieren. Diese Optimierung existierte im EIRENE Code bislang nur für volumengemittelte Größen (seit ca. 1990), und soll im folgenden Kapitel dargestellt werden.

6. Optimierung der Berechnung der Standardabweichung

Der EIRENE Code ist ein Monte Carlo Code, der die von ihm gelieferten Ergebnisse mit Hilfe von statistischen Methoden aus den während der Flugzeit der Teilchen gesammelten Informationen ermittelt. Um die Güte der gelieferten Resultate beurteilen zu können, werden zu allen Ergebnisgrößen Standardabweichungen berechnet. Für die Berechnung der Standardabweichung für flächengemittelte Größen wurden bislang für jede Größe Schleifen über alle Flächen und alle Teilchensorten verwendet, um die entlang der Teilchenbahn gesammelten Informationen zu verarbeiten. Bei den bisherigen Rechnungen mit eher kleinen Flächenzahlen, war der Aufwand hierfür vertretbar. Bei Geometrien mit mehreren Tausend Flächen hingegen ist der Aufwand beträchtlich, insbesondere, da die meisten Teilchen jeweils nur einen Bruchteil aller Flächen treffen.

Die nun durchgeführte Optimierung dieses Programmteils geschah analog zu der Optimierung der Berechnung der Standardabweichung für volumengemittelte Größen, die schon vor längerer Zeit implementiert wurde. Dabei wird während des Teilchenflugs eine Liste mit den Nummern der getroffenen Flächen aufgebaut (bei den volumengemittelten Größen war dies eine Liste der aktuell von einem Testteilchen wirklich durchflogenen Zelle.)

Zusätzlich wird registriert, welche Teilchensorte die jeweilige Fläche getroffen hat. Nur die getroffenen Flächen werden anschließend in der Berechnung des Beitrags des Teilchens zu der Standardabweichung berücksichtigt. Die entsprechenden Listen werden für jedes neue Testteilchen neu initialisiert. Zusätzlich werden auch nur Standardabweichungen bearbeitet, die zu Ergebnisgrößen des entsprechende Teilchentyps gehören. Wenn während einer Teilchenhistorie z.B. eine Fläche nur von Atomen getroffen wurde, müssen die Standardabweichungen für Molekülflüsse für diese Historie nicht bearbeitet werden.

Die Optimierung des Codes verbessert die Laufzeit des EIRENE-Codes für große Geometrien trotz der Berechnung von Standardabweichung der Ergebnisgrößen. Eine Laufzeitreduktion um 85% und mehr ist gegeben, nun ähnlich den Einsparungen bei den Zylindern, die ohne aktive Berechnung der Standardabweichung als Beispiele für den Octree simuliert worden sind.

Einsparungen von nur 50% bei großen Geometrien ohne Optimierung des Statistik-Codes - siehe Abschnitt 5.2 - sind erklärbar durch den hohen Anteil der Statistikberechnung an der Gesamtlaufzeit. Diesen Zusammenhang verdeutlicht die Abbildung 6.1. Die oben beschriebene Optimierung des Codes führt - im konkreten Fall der Portplug-Geometrien vorliegend - zu einem verschwindend geringen Aufwand.

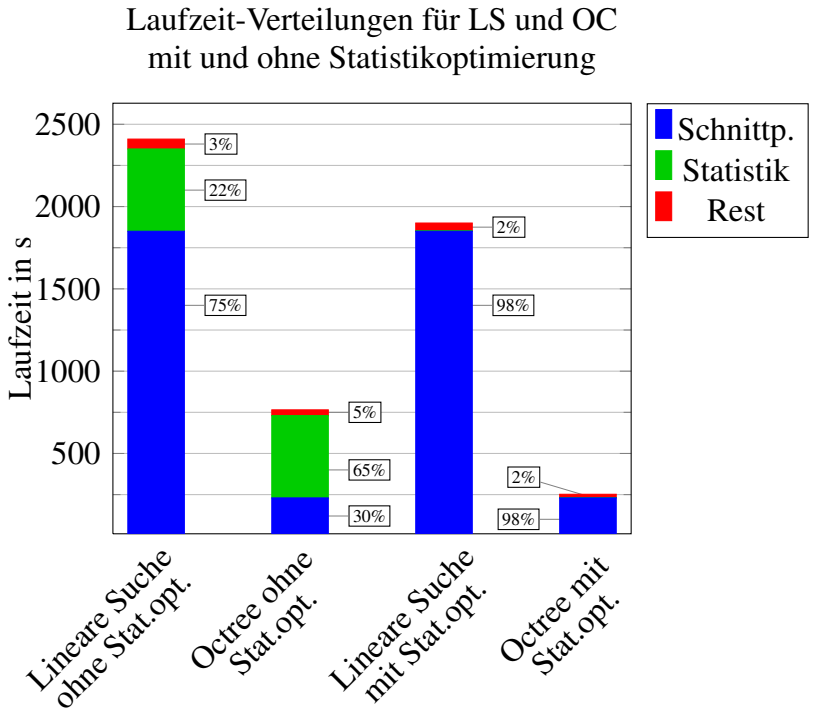


Abbildung 6.1.: Verteilung der Rechenzeit für die Simulation der Geometrie M1. Die Werte sind den Ergebnissen des Profilings mittels GNU gprof entnommen.

7. Fazit und Ausblick

Im Zuge der vorliegenden Arbeit ist eine Erweiterung des EIRENE-Codes entstanden, die erlaubt, deutlich komplexere geometrische Anordnungen in der Simulation zu berücksichtigen, bei gleichbleibender oder sogar reduzierter Laufzeit des Codes. Je größer die Komplexität des Models, desto größer ist der Laufzeitgewinn durch die Optimierung gegenüber dem ursprünglichen Code. Als Beispiel dafür ist die Simulation von Port-Plugs für den Fusionsreaktor ITER genannt worden, die zuvor nur unter wesentlich höherem Zeitaufwand durchgeführt werden konnte. Grundlegende Idee dazu war, Techniken und Strategien zur Beschleunigung von Raytracing bei der Bildverarbeitung für die mathematisch ähnlich gelagerte Problematik der Monte-Carlo-Simulation von Teilchentransport einzusetzen. Es folgte die Entscheidungsfindung zur Auswahl der geeignetsten Strategie; die Technik der Octrees ist als geeignete Strategie identifiziert worden. Die Implementierung eines Octrees war die Hauptaufgabe dieses hier beschriebenen Projekts.

Es wurde gezeigt, dass auch bei Verwendung des erweiterten Programmes die Ergebnisse der ursprünglichen Simulationen exakt reproduzierbar sind. Für Geometrien mit gleichmäßig verteilten Flächen sind die Laufzeiten um einen Faktor 5 bis ≥ 20 gesunken. Eine obere Grenze ist nicht anzugeben, da die lineare Suche in den Fällen CYLINDER.P24 und CYLINDER.P48 nach 7200 Sekunden aufgrund der Laufzeitbeschränkung abgebrochen worden ist. Die gemittelte relative Laufzeitersparnis liegt zwischen 80% und $\geq 94\%$. Die Fälle M1, M1BAF und M1M4 sind mit einer Laufzeitreduktion von im Mittel einem Faktor 2 bzw. 50% ebenso deutlich schneller berechenbar. Unter Ausnutzung der optimierten Statistikberechnung ist dieser Faktor deutlich höher (Faktor 10-15), jedoch ohne weitere Laufzeittests vorerst nicht quantitativ sicher belegbar.

Die Nutzbarkeit des Octrees bei Einsatz von verteiltem Rechnen („parallel computing“) wurde sichergestellt. Für den Nutzer ändert sich in der Nutzung von EIRENE nur die Aktivierung der Erweiterung in der Eingabe, alles Weitere ist für ihn transparent. Die Implementierung des Octrees erlaubt die Nutzung der linearen Suche durch einfache Deaktivierung mittels eines logischen Schalters in der Eingabe. Aufgrund der Kapselung der Schnittpunkt-Berechnung in der Routine `CheckInter` kann der potentiell nicht seiteneffektfreie Code weiter verwendet werden.

Die Möglichkeit zur 3D-Darstellung mit EGView3D erlaubt die Visualisierung von Prozessen und Flugbahnen. Der EIRENE-Code wurde dazu mit der Möglichkeit zur Ausgabe der Drei-, Vier- und Fünfecke sowie der Octree-Blöcke und Flugbahnen erweitert.

Als Schwäche der momentanen Implementierung wurde aufgezeigt, dass die Beschaffenheit der Geometrie und die Flächenverteilung noch zu starken Einfluss auf das Laufzeitverhalten hat. Es ist daher als Option für die Zukunft anzusehen, diese Abhängigkeit aufzulösen.

Dazu bietet sich die angesprochene Unterteilung am Objektmittelpunkt an, jedoch mit dem Nachteil, dass die Location Codes nicht mehr wie von [1] vorgestellt nutzbar sind. Es ist an dieser Stelle in Zukunft abzuwägen und zu testen, ob eine andere Aufteilung oder die Gestattung tieferer Bäume weitere Zeitersparnisse ermöglichen. Für die gleichförmig gestalteten Geometrien bei Zylindern sind die Ergebnisse bereits als gut zu bewerten.

Ebenso ist die Schwäche in Bezug auf die in der Eingabe mögliche Definition von Löchern in Flächen zu beheben. Eine Erweiterung des neuen Codes führt zur Berücksichtigung von Flächen 2. Ordnung im Octree. Ein leicht erhöhter Speicherbedarf durch den Octree stellt bislang kein Problem dar.

Dennoch bleibt die Frage, ob das Speichern der Indizes in den Knoten besser mit Listen zu lösen ist, um die Über-Allokation für Indizes der Flächen eines Elterknotens, die nicht im jeweiligen Kind liegen, zu vermeiden. Dagegen sprach bisher, dass mit einfachen Feldern keine aufwändige Verwaltung von Zeigern notwendig ist.

Insgesamt leistet die jetzige Implementierung das Geforderte, jedoch müssen mit weiteren, möglichst unregelmäßigen Geometrien mit 50.000, 100.000 und mehr Dreiecken ebenfalls Testläufe absolviert werden. Bisher existieren solche Modell-Geometrien jedoch nicht. Eine Möglichkeit ist die Zweckentfremdung von großen, geschlossenen Modellen aus der Computergrafik, da die dort nicht gegebene physikalische Relevanz dieser Modelle für Laufzeitanalysen des Octree-Codes unerheblich ist. Die Simulation mit aktivierten Stoßprozessen im Volumen (also nicht nur an den berechneten Schnittpunkten auf den Oberflächen) sind ebenso nur exemplarisch Teil der Arbeit gewesen.

Die bisherige Implementierung ist hinreichend getestet, für den Praxiseinsatz geeignet und ermöglicht bereits in dieser Phase der Entwicklung eine mindestens 50 bis 80% schnellere Simulation.

A. EGView3D-Erweiterung

Zur Analyse von Problemen während der Tests wurde das Javaprogramm EGView3D, das im Rahmen von [3] initiiert wurde, erweitert. Das Werkzeug dient der Visualisierung von EIRENE-Geometrien und ist primär in der Lage, „Additional Surfaces“ als 3D-Bild darzustellen.

Ursprünglich war die Software darauf ausgelegt, Flächen 1. und 2. Ordnung darzustellen, die mittels der aus Abschnitt 4.1 bekannten Koeffizienten $a_0, \ldots, a_9$ der impliziten Flächengleichung definiert sind. EIRENE besitzt für die Ausgabe der Flächen ein Modul, das die Flächen in Form einer XML-Datei ausgibt. Diese Datei wird durch den Benutzer in EGView3D ausgewählt und durch das Programm verarbeitet. Die Ausgabe der Datei wird durch den logischen Schalter `PLTADD` in Block 11 der Eingabedatei aktiviert (siehe EIRENE-Handbuch [2]).

Bei der Verarbeitung werden mit Hilfe des „Visualisation Toolkits“ (VTK[1]) die XML-Daten in eine 3D-Grafik gewandelt, die dem Nutzer auf dem Bildschirm präsentiert wird und die er in eine Bilddatei exportieren kann. Die Darstellung kann zwischen Drahtgittermodell und gefüllten („solid“) Flächen wechseln.

Die Erweiterung des Ausgabemoduls und des Programms erlaubt die Darstellung der Drei-, Vier- und Fünfecke in der 3D-Grafik. Diese werden nicht über den Umweg der Ebenengleichungen dargestellt, sondern mittels ihrer Eckpunkte angegeben. Die Ausgabe erfolgt unabhängig davon, ob mit Octree gerechnet wird oder nicht. Sollte der Octree und ein weiterer logischer Schalter `TRCOC` in Block 11 der EIRENE-Eingabe aktiv sein, werden von EIRENE außerdem die Eckpunkte der Blätter des Octrees zur Visualisierung in die XML-Datei geschrieben. Es ist somit möglich, Octree-Aufbau und „Additional Surfaces” zusammen zu betrachten.

Ebenfalls ist die Möglichkeit zur Darstellung einer Teilchenverfolgung hinzu gekommen. Ist `PLTADD` aktiviert, werden für jedes Teilchen getrennt und unabhängig von der Nutzung eines Octree, alle Schnittpunkte mit Flächen der Geometrie ausgegeben. Diese Daten können in EGView3D als Zusatzdatei angegeben werden, um die Flugbahn eines Teilchens von Anfang bis Ende der Verfolgung darzustellen.

Das Werkzeug wurde in seiner erweiterten Form genutzt, Fehler bei der Nutzung des Octrees aufzudecken, die aufgrund von abweichenden Ergebnissen im Vergleich zur linearen Suche aufgefallen sind. Die Möglichkeit, Geometrie, Octree-Blätter und Teilchenflugbahn zu visualisieren, erleichtert die Fehlersuche, da angrenzende Flächen und Blöcke grafisch einfacher zu erkennen sind als in der Form von Zahlen. Mit EGView3D erzeugte Beispielgrafiken sind die in Anhang D dargestellten 3D-Plots der Testgeometrien.

[1] The Visualisation ToolKit: `http://www.vtk.org`

B. Datenstrukturen des Octrees

Variable	Beschreibung	siehe Abschnitt
root	Zeiger auf den Wurzelknoten	
layers	Höhe des Baumes	4.2.1
bounds	3x2 Matrix mit den erweiterten Eckpunkten $\tilde{B}_1, \tilde{B}_2$	4.2.1
length	Kantenlängen des Boundary Volumes von $\tilde{B}_1, \tilde{B}_2$	4.2.1
maxvalue	$= 2^{layers-1}$	4.3.1
shortest	Länge der kürzesten Kante eines Volumens	4.2.2, 4.3.1
factor	Divisor von shortest	4.2.2, 4.3.1

Tabelle B.1.: Typ `ocTree` zur Speicherung des Octrees.
Nähere Erläuterungen in den genannten Abschnitten.

Variable	Beschreibung	siehe Abschnitt
number	Integer-Feld zur Speicherung des „Location Codes“	4.2.3
layer	Ebene, auf der sich der Knoten befindet	4.2.2
B	3x3-Matrix mit 2 Eckpunkten und Raummittelpunkt	4.2.2
parent	Zeiger auf den Elternknoten	
children	Feld mit Zeigern auf die Kinder	4.2.2
radius	Radius der Umkugel des Knotenvolumens	4.2.4
surfaces	Feld mit den Flächenindizes	4.2.4
nsurfaces	Anzahl der assoziierten Flächen	4.2.4

Tabelle B.2.: Typ `ocNode` zur Speicherung eines Knotens des Octrees.
Nähere Erläuterungen in den genannten Abschnitten.

C. Ergänzende Grafiken und Tabellen zur Laufzeitanalyse

Rechner	ipp395	ipp960
CPU-Generation	Intel Core2	Intel Core2
CPU-Typ	Quadcore	Quadcore
CPU-Geschw.	2,66Ghz	2,66Ghz
RAM	8096MB	8096MB
System	Linux-Kernel 2.6.34	Linux-Kernel 2.6.34
Compiler	GNU gfortran 4.5.0	Intel ifort 12.0.4

Tabelle C.1.: Konfiguration der Computer zur statistischen Messdatenerfassung

Fall:	CYLINDER.P12				CYLINDER.P12.PLASMA			
Teilchen	lin. Suche in [s]		Octree in [s]		lin. Suche in [s]		Octree in [s]	
Comp.-Opt.:	nein	ja	nein	ja	nein	ja	nein	ja
5.000	56	41	10	7	22	18	4	3
10.000	107	77	20	12	41	36	9	7
50.000	516	385	92	62	263	178	47	32
100.000	1121	782	193	114	546	377	99	61
Reduktion:			82,31%	84,30%			80,32%	82,20%

Fall:	CYLINDER.P24				CYLINDER.P24.PLASMA			
Teilchen	lin. Suche in [s]		Octree in [s]		lin. Suche in [s]		Octree in [s]	
Comp.-Opt.:	nein	ja	nein	ja	nein	ja	nein	ja
5.000	422	307	29	20	231	184	15	10
10.000	803	613	56	39	441	366	29	20
50.000	4167	3219	277	196	2238	1819	135	99
100.000	7201	6827	553	393	3627	3607	268	195
Reduktion:			92,96%	93,84%			93,40%	94,52%

Fall:	CYLINDER.P48				CYLINDER.EIR			
Teilchen	lin. Suche in [s]		Octree in [s]		lin. Suche in [s]		Octree in [s]	
Comp.-Opt.:	nein	ja	nein	ja	nein	ja	nein	ja
5.000	2539	1724	102	69	2	2	2	2
10.000	4448	3589	147	94	4	3	4	3
50.000	7213	7210	650	426	20	16	20	17
100.000	7212	7210	1236	846	41	33	40	33
Reduktion:			91,63%	93,93%			-	-

Fall:	M1				M1BAF			
Teilchen	lin. Suche in [s]		Octree in [s]		lin. Suche in [s]		Octree in [s]	
Comp.-Opt.:	nein	ja	nein	ja	nein	ja	nein	ja
5.000	25	13	16	6	50	22	31	9
10.000	51	26	31	10	83	43	46	16
50.000	248	130	124	47	387	209	252	81
100.000	483	266	244	85	800	415	322	162
Reduktion:			43,78%	63,35%			44,53%	60,64%

Fall:	M1M4			
Teilchen	lin. Suche in [s]		Octree in [s]	
Comp.-Opt.:	nein	ja	nein	ja
5.000	36	29	26	16
10.000	69	57	48	29
50.000	322	269	217	128
100.000	674	549	428	268
Reduktion:			31,88%	49,05%

Tabelle C.2.: Laufzeitstatistik für die Testgeometrien, Compiler GNU gfortran. Reduktion über Laufzeiten aller Teilchenverfolgungen gemittelt. Laufzeit über 10 Läufe auf ganze Sekunden gemittelt, jeder mit neuen Zufallszahlen.

Fall:	CYLINDER.P12				CYLINDER.P12.PLASMA			
Teilchen	lin. Suche in [s]		Octree in [s]		lin. Suche in [s]		Octree in [s]	
Comp.-Opt.:	nein	ja	nein	ja	nein	ja	nein	ja
5.000	80	38	17	5	37	30	8	3
10.000	148	77	33	9	77	59	15	6
50.000	643	383	165	44	334	177	79	22
100.000	1313	797	311	81	686	362	145	45
Reduktion:			76,79%	88,89%			79,06%	89,07%

Fall:	CYLINDER.P24				CYLINDER.P24.PLASMA			
Teilchen	lin. Suche in [s]		Octree in [s]		lin. Suche in [s]		Octree in [s]	
Comp.-Opt.:	nein	ja	nein	ja	nein	ja	nein	ja
5.000	506	385	49	16	280	191	29	7
10.000	1025	661	94	28	526	409	55	14
50.000	4979	3249	433	128	2730	2014	266	70
100.000	7201	7093	866	248	5453	4075	504	138
Reduktion:			90,12%	96,05%			90,09%	96,47%

Fall:	CYLINDER.P48				CYLINDER.EIR			
Teilchen	lin. Suche in [s]		Octree in [s]		lin. Suche in [s]		Octree in [s]	
Comp.-Opt.:	nein	ja	nein	ja	nein	ja	nein	ja
5.000	2522	2062	181	75	2	1	2	1
10.000	5031	4126	315	101	3	2	3	2
50.000	7239	7245	1215	319	15	7	15	7
100.000	7233	7246	2373	571	29	14	29	14
Reduktion:			84,25%	95,41%			-	-

Fall:	M1				M1BAF			
Teilchen	lin. Suche in [s]		Octree in [s]		lin. Suche in [s]		Octree in [s]	
Comp.-Opt.:	nein	ja	nein	ja	nein	ja	nein	ja
5.000	28	14	17	7	61	23	41	16
10.000	62	29	41	12	102	51	59	32
50.000	291	143	164	56	445	241	317	148
100.000	561	288	415	122	923	431	511	230
Reduktion:			36,14%	57,83%			37,41%	38,31%

Fall:	M1M4			
Teilchen	lin. Suche in [s]		Octree in [s]	
Comp.-Opt.:	nein	ja	nein	ja
5.000	43	36	35	19
10.000	86	65	68	44
50.000	397	337	310	234
100.000	793	673	692	454
Reduktion:			18,05%	35,18%

Tabelle C.3.: Laufzeitstatistik für die Testgeometrien, Compiler Intel ifort. Reduktion über Laufzeiten aller Teilchenverfolgungen gemittelt. Laufzeit über 10 Läufe auf ganze Sekunden gemittelt, jeder mit neuen Zufallszahlen.

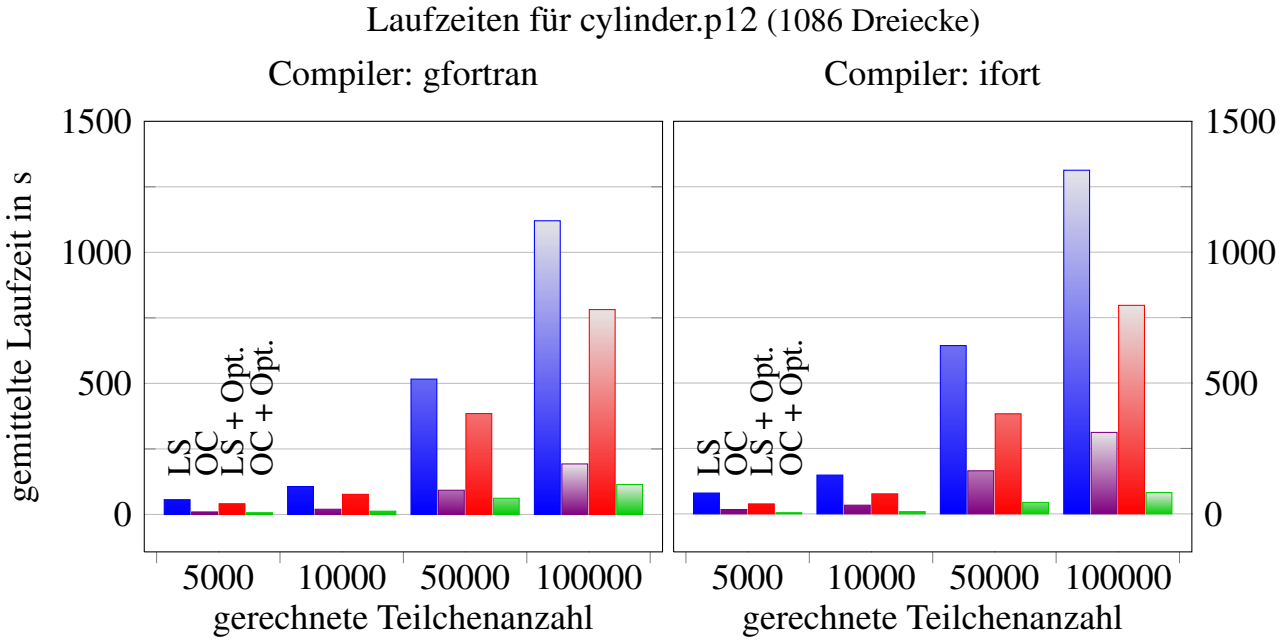


Abbildung C.1.: Laufzeit-Analyse für Fall CYLINDER.P12, 1086 Dreiecke.
Datenbasis: Tabellen C.2 und C.3.
LS: Lineare Suche, OC: Octree, jeweils mit und ohne Optimierung durch den Compiler

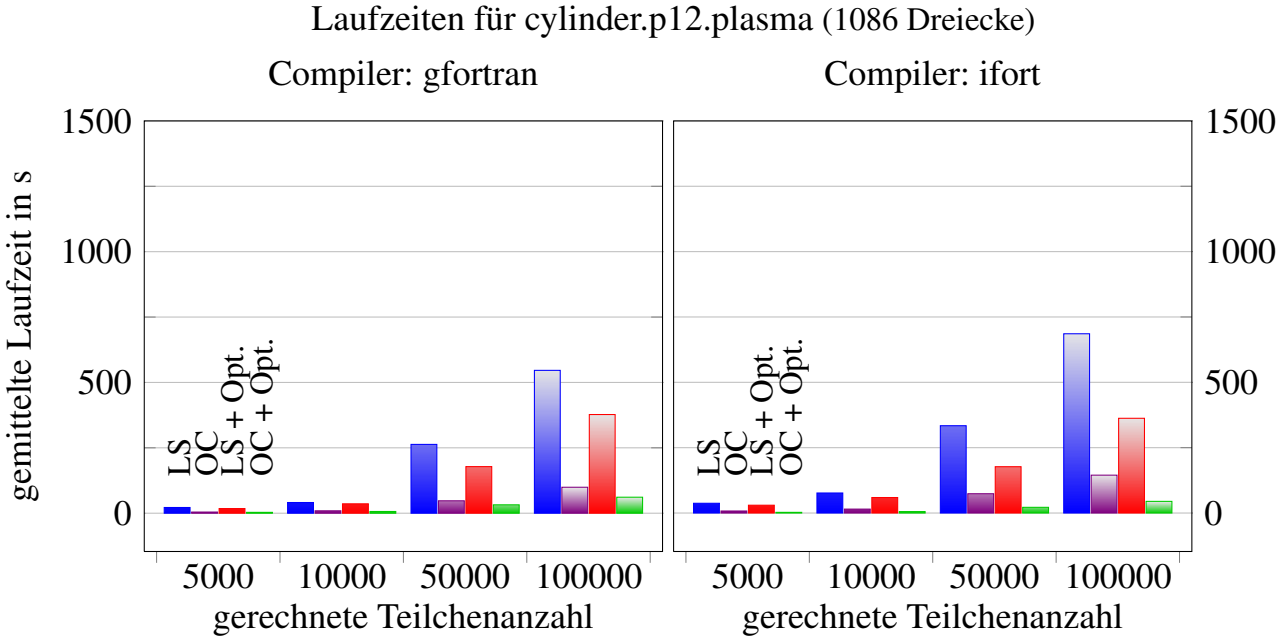


Abbildung C.2.: Laufzeit-Analyse für Fall CYLINDER.P12.PLASMA, 1086 Dreiecke und Plasma im Inneren.
Statistische Datenbasis und Darstellung wie Abb. C.1

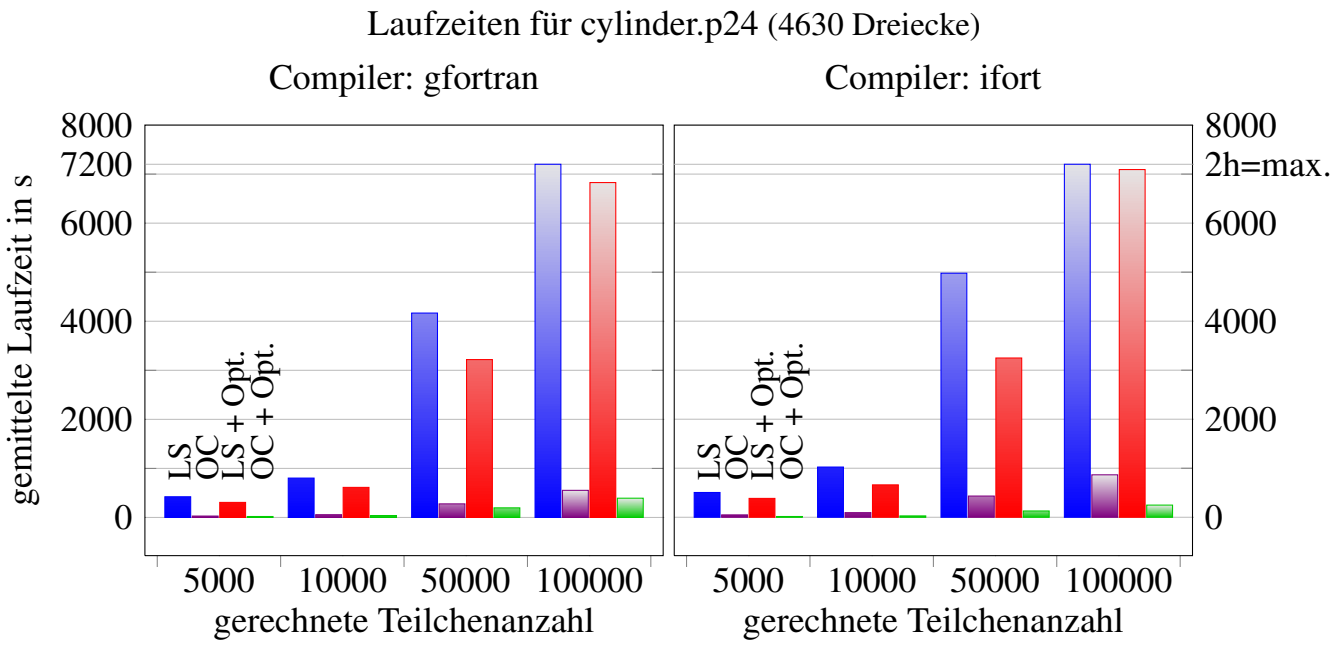


Abbildung C.3.: Laufzeit-Analyse für Fall CYLINDER.P24, 4630 Dreiecke.
Statistische Datenbasis und Darstellung wie Abb. C.1

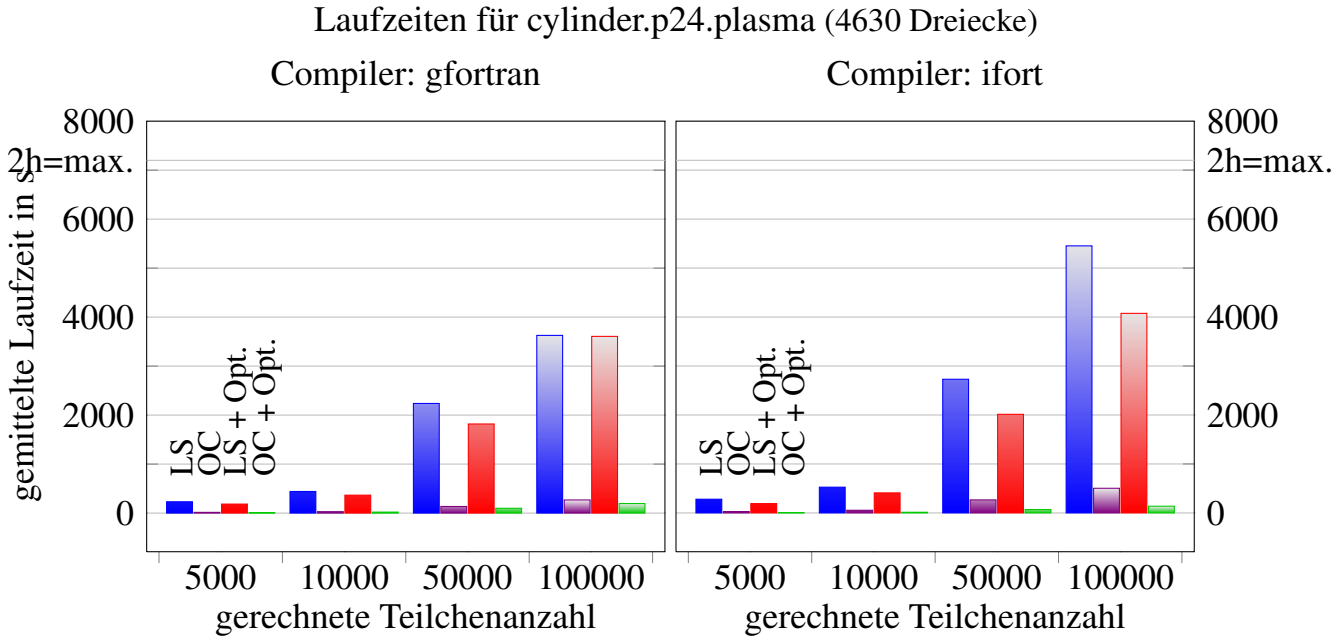


Abbildung C.4.: Laufzeit-Analyse für Fall CYLINDER.P24.PLASMA, 4630 Dreiecke und Plasma im Inneren.
Statistische Datenbasis und Darstellung wie Abb. C.1

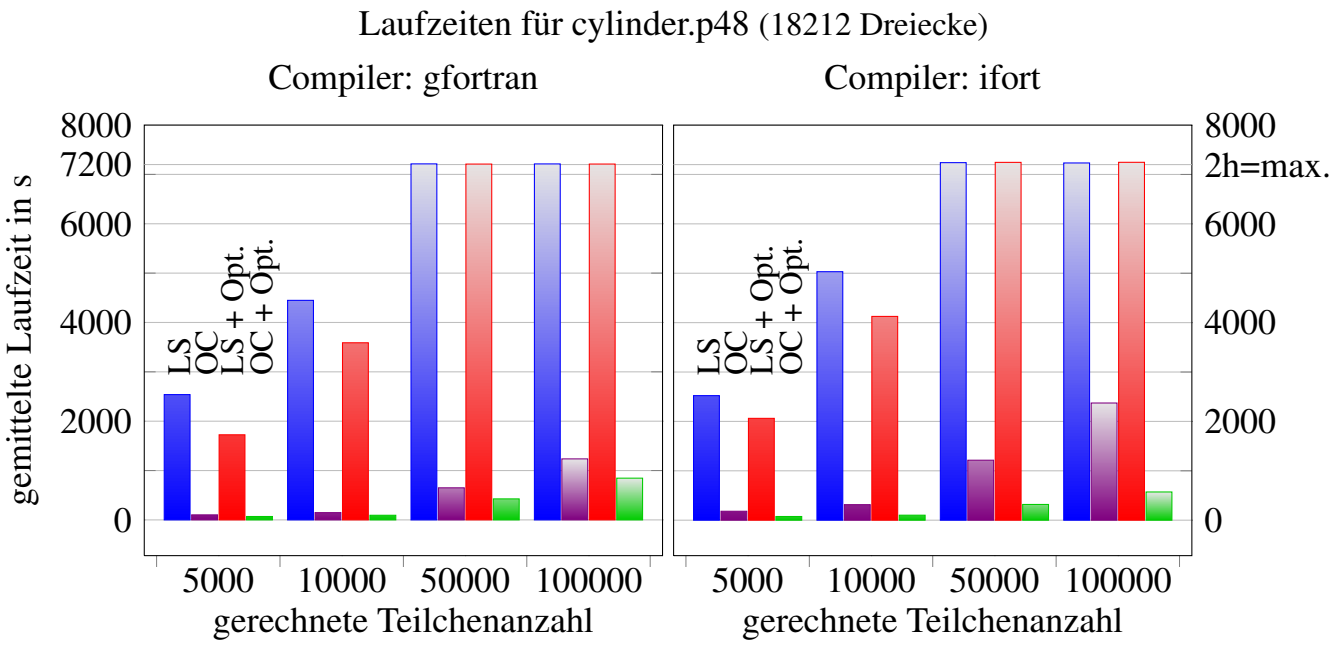


Abbildung C.5.: Laufzeit-Analyse für Fall CYLINDER.P48, 18212 Dreiecke.
Statistische Datenbasis und Darstellung wie Abb. C.1

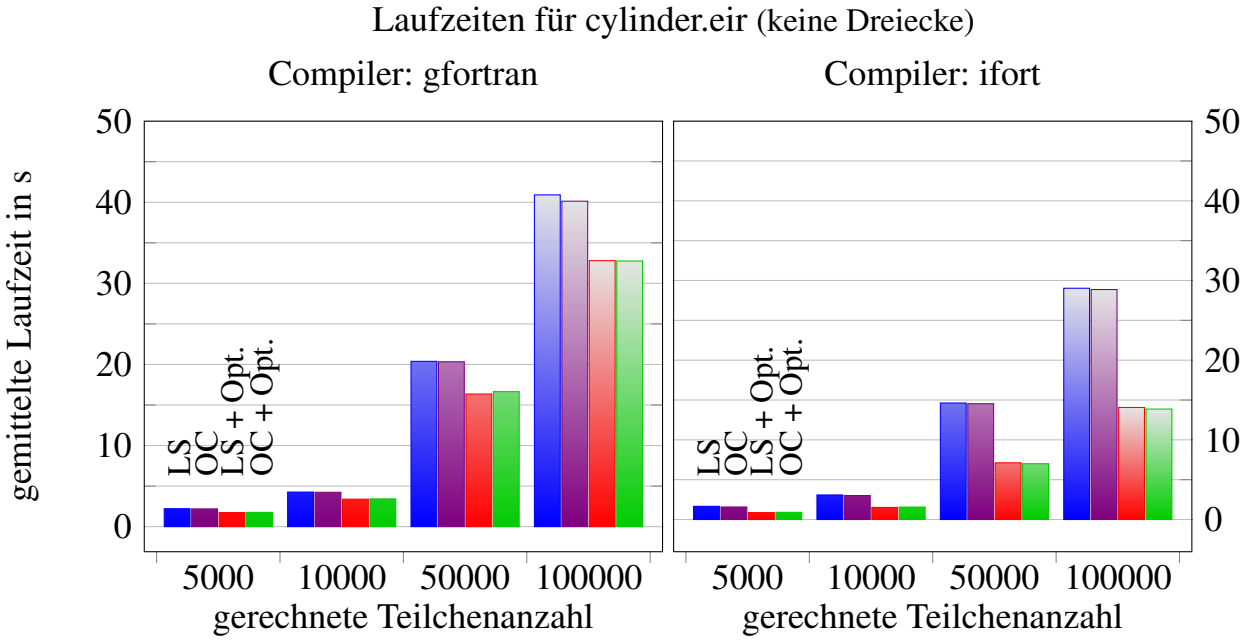


Abbildung C.6.: Laufzeit-Analyse für Fall CYLINDER.EIR, nur Flächen 1. und 2. Ordnung.
Statistische Datenbasis und Darstellung wie Abb. C.1

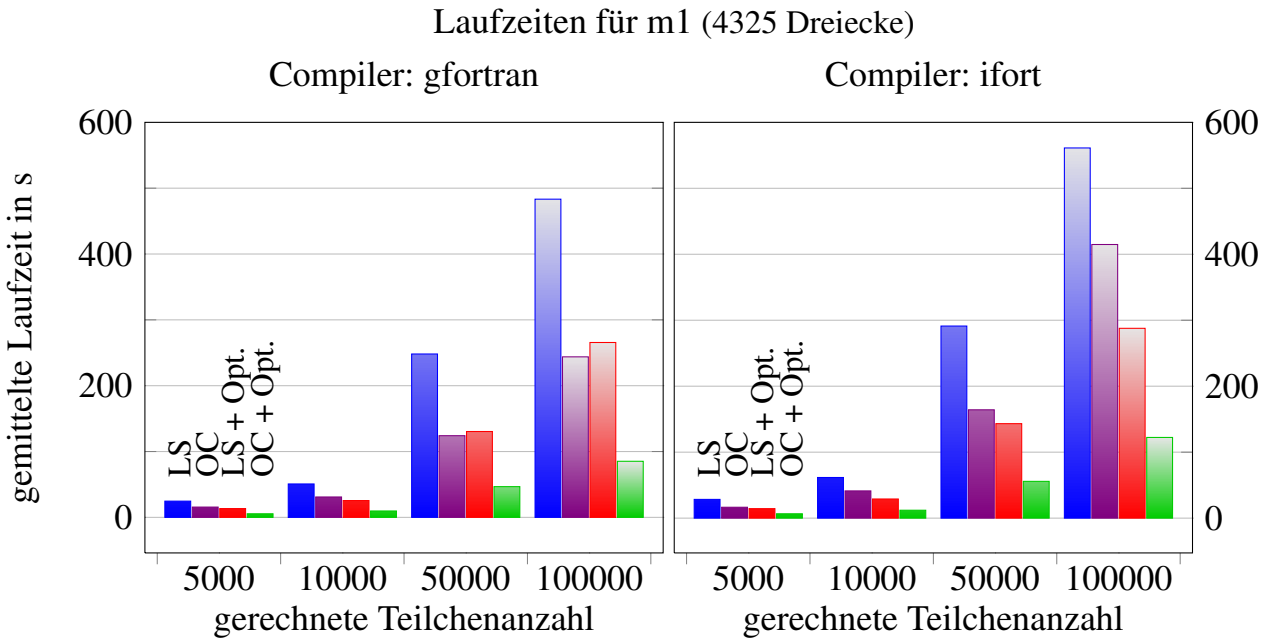


Abbildung C.7.: Laufzeit-Analyse für Fall M1, 4325 Dreiecke.
Statistische Datenbasis und Darstellung wie Abb. C.1

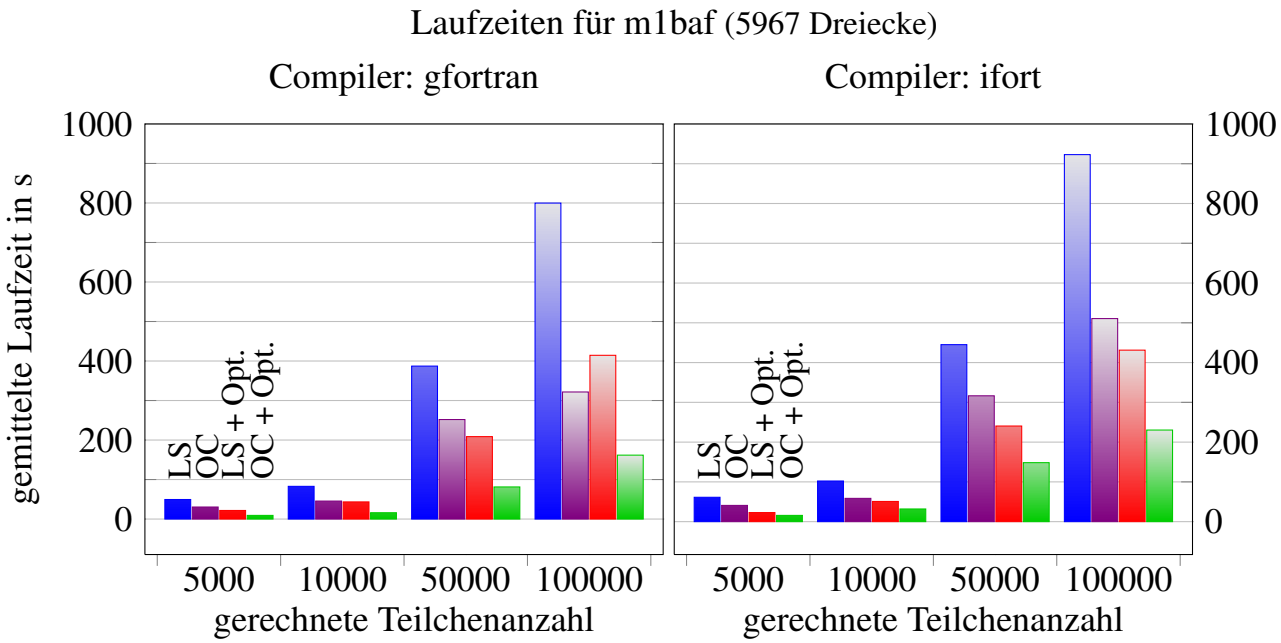


Abbildung C.8.: Laufzeit-Analyse für Fall M1BAF, 5967 Dreiecke.
Statistische Datenbasis und Darstellung wie Abb. C.1

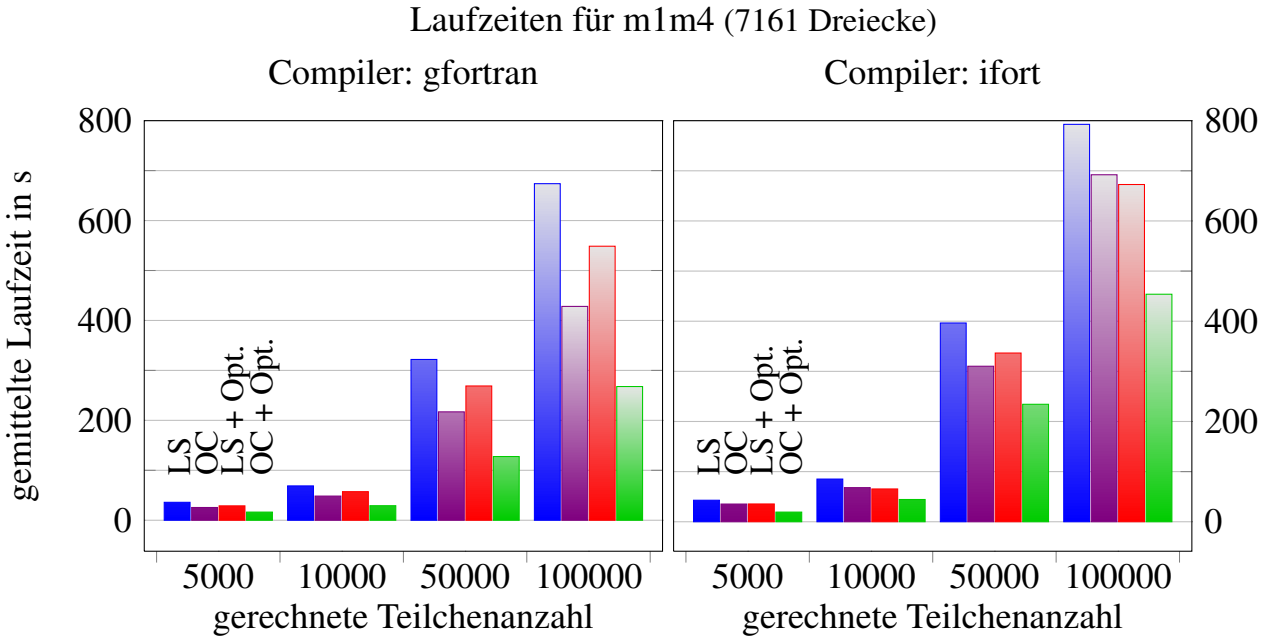


Abbildung C.9.: Laufzeit-Analyse für Fall M1M4, 7161 Dreiecke.
Statistische Datenbasis und Darstellung wie Abb. C.1

D. 3D-Plots der Testgeometrien

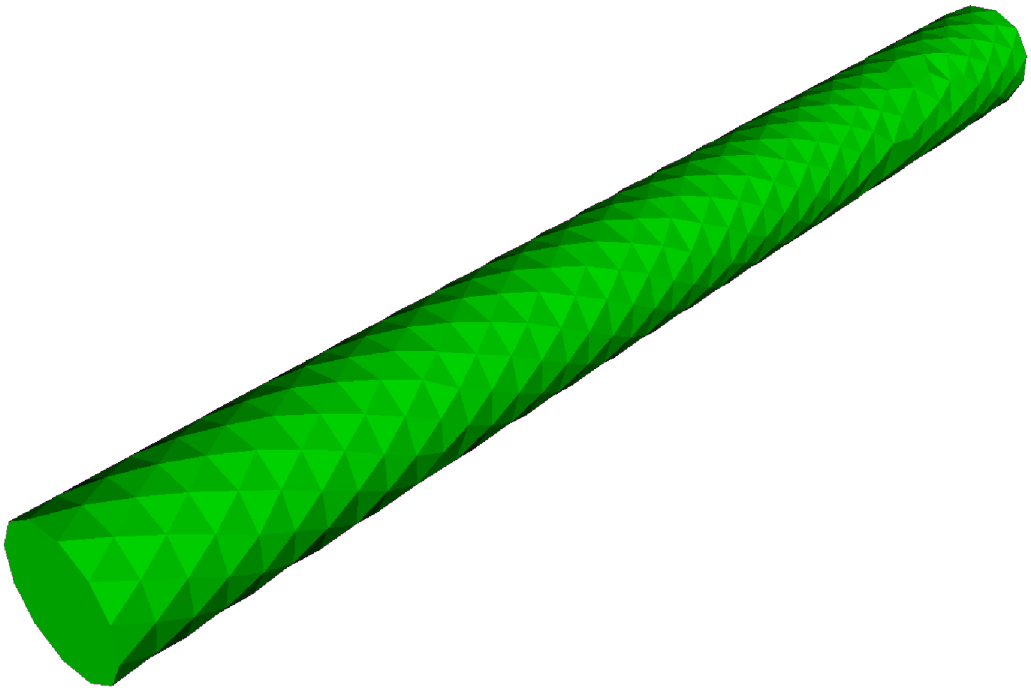


Abbildung D.1.: 3D-Plot der Geometrie von CYLINDER.P12 und CYLINDER.P12.PLASMA, 1086 Dreiecke

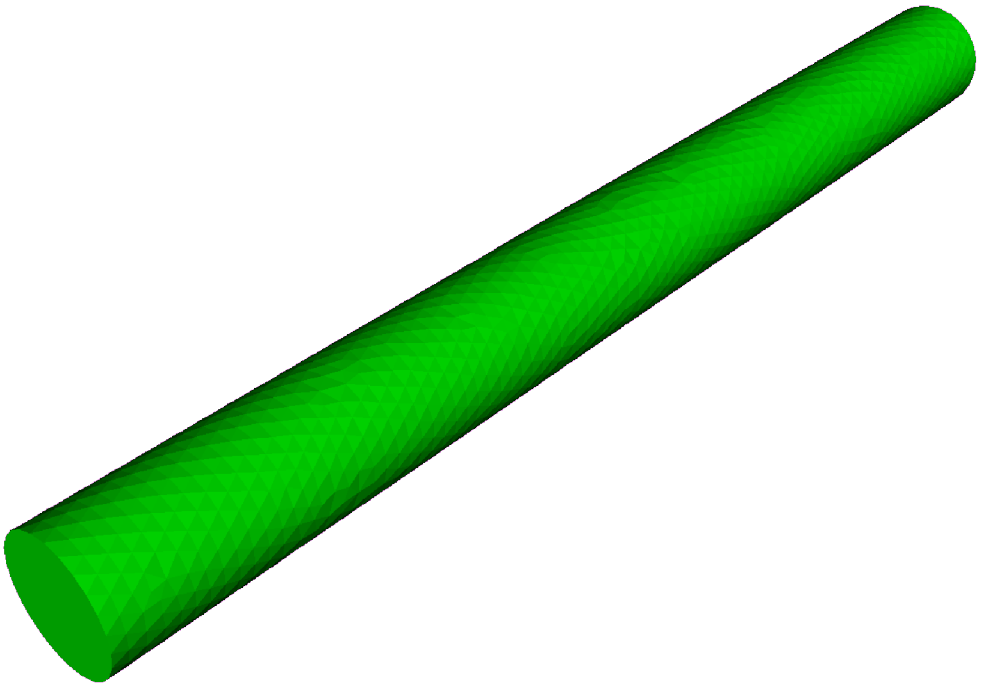


Abbildung D.2.: 3D-Plot der Geometrie von CYLINDER.P24 und CYLINDER.P24.PLASMA, 4630 Dreiecke

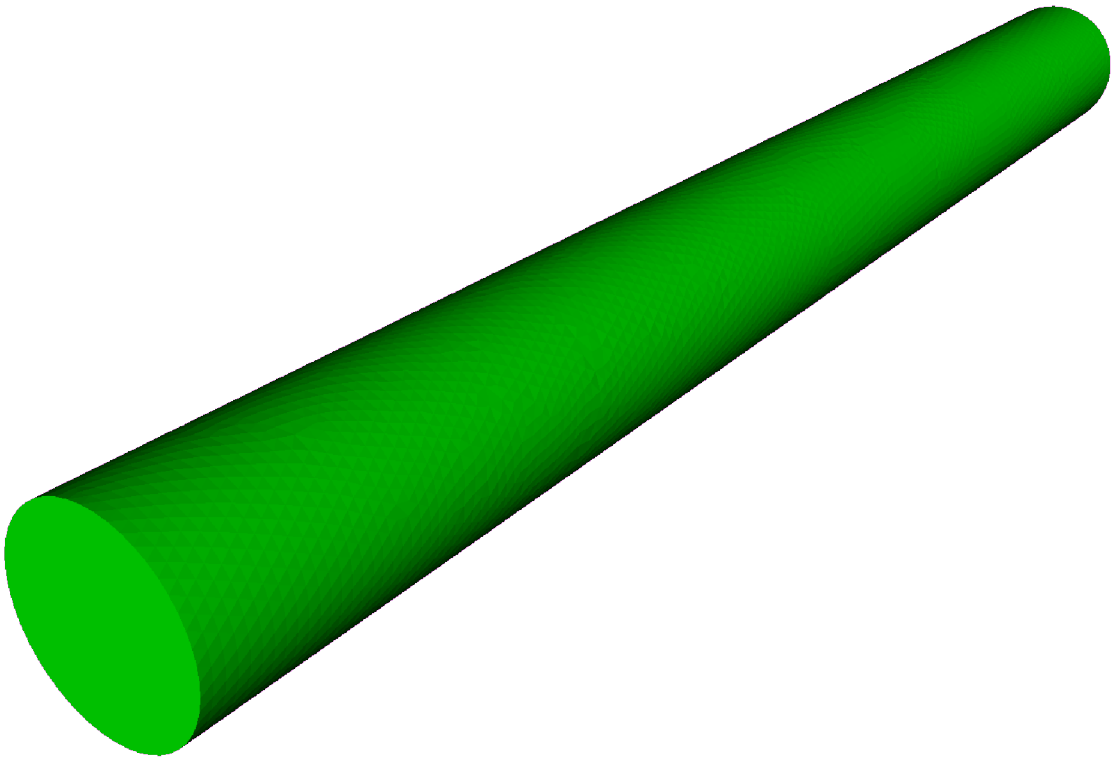


Abbildung D.3.: 3D-Plot der Geometrie von CYLINDER.P48, 18212 Dreiecke

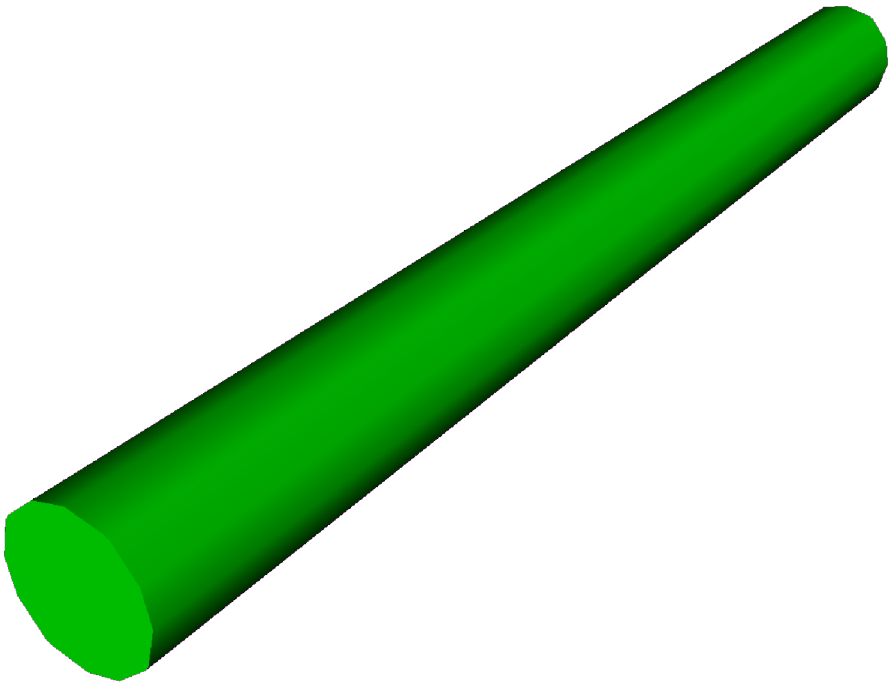


Abbildung D.4.: 3D-Plot der Geometrie von CYLINDER.EIR, nur Flächen 1. und 2. Ordnung.
Die Stirnseiten sind nicht exakt rund, da die Flächen intern von VTK zur Darstellung in Dreiecke zerlegt werden.

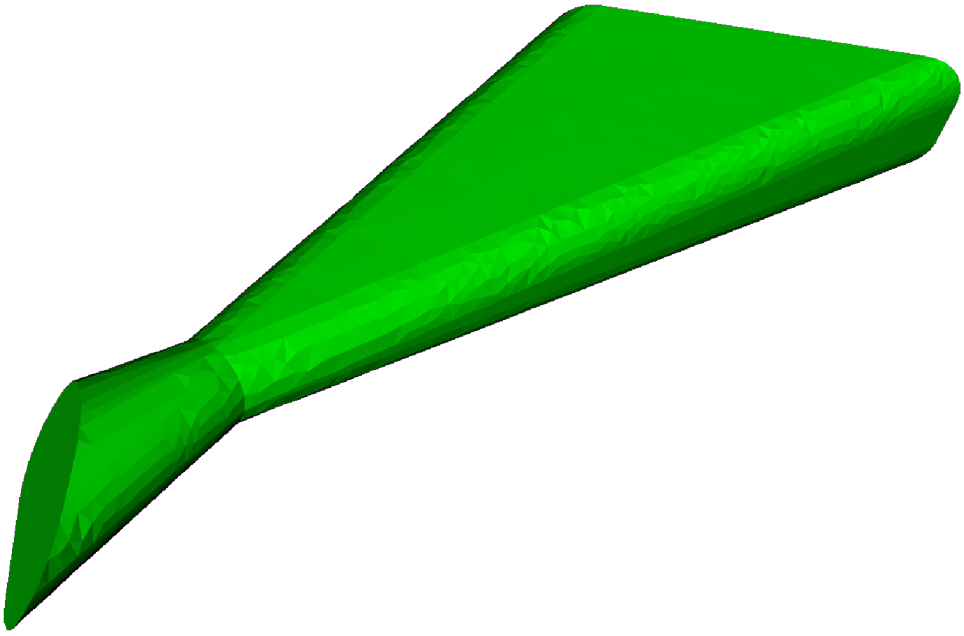


Abbildung D.5.: 3D-Plot des Geometrie von M1, 4325 Dreiecke

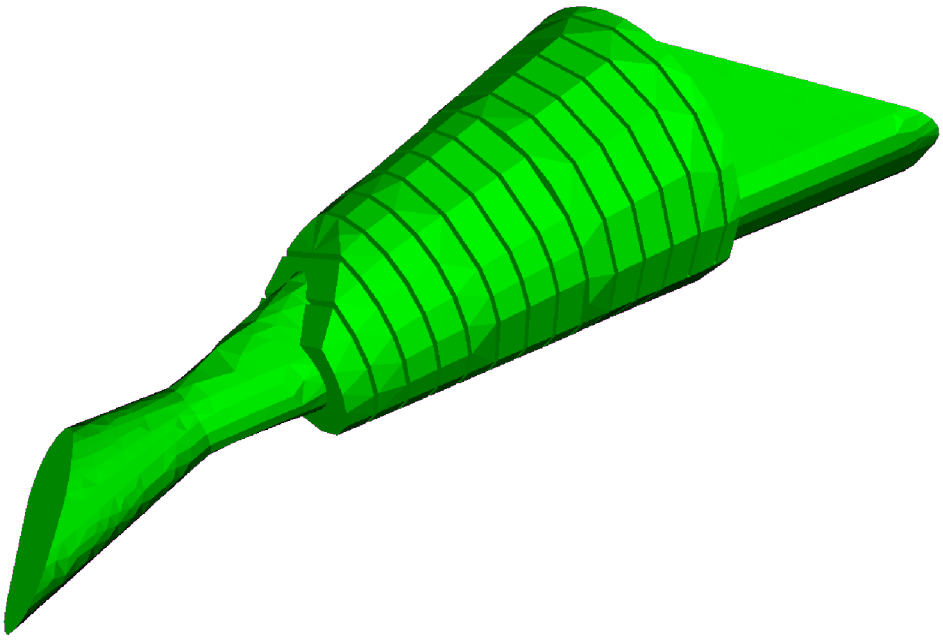


Abbildung D.6.: 3D-Plot des Geometrie von M1BAF, 5967 Dreiecke

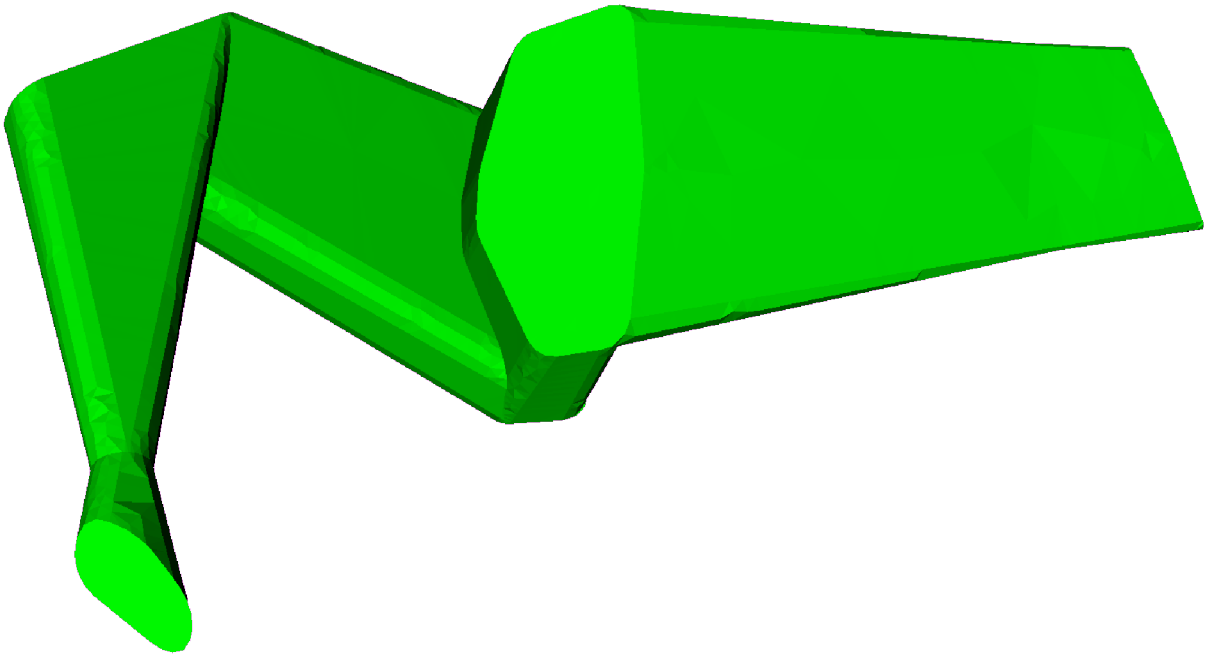


Abbildung D.7.: 3D-Plot der Geometrie von M1M4, 7161 Dreiecke

Literaturverzeichnis

[1] S. Frisken et al. *Simple and Efficient Traversal Methods for Quadtrees and Octrees*. Journal of Graphics Tools, Bd. 7, Nr. 3 (2003).

[2] D. Reiter. *The EIRENE Code User Manual, http://www.eirene.de*. Forschungszentrum Jülich (2009).

[3] A. Steinheuer. *Implementierung eines Konverters und einer Visualisierung für kombinatorische Beschreibungen von 3D Konfigurationen in Monte Carlo Codes*. Diplomarbeit, Fachhochschule Aachen (2009).

[4] ISO 10303:2010. *Automation systems and integration - Product data representation and exchange* (2010).

[5] A. Chang. *A survey of geometric data structures for ray tracing*. Techn. Ber., Polytechnic University (New York) (2001).

[6] T. G. Pfeiffer. *Raytracing* (2007).

[7] A. Appel. *Some techniques for shading machine renderings of solids*. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, S. 37–45. ACM (1968).

[8] T. Whitted. *An Improved Illumination Model for Shaded Display.* Commun. ACM, Bd. 23, Nr. 6:343–349 (1980).

[9] T. Ize et al. *Grid creation strategies for efficient ray tracing*. In *Interactive Ray Tracing, 2007. RT'07. IEEE Symposium on*, S. 27–32. IEEE (2007).

[10] A. Lagae et al. *Compact, Fast and Robust Grids for Ray Tracing.* Comput. Graph. Forum, Bd. 27, Nr. 4:1235–1244 (2008).

[11] H. Fuchs et al. *On visible surface generation by a priori tree structures*. In *ACM Siggraph Computer Graphics*, Bd. 14, S. 124–133. ACM (1980).

[12] M. De Berg et al. *Computational geometry: algorithms and applications*. Springer-Verlag New York Inc (2008).

[13] I. Wald et al. *On building fast kd-trees for ray tracing, and on doing that in O (N log N)*. In *Interactive Ray Tracing 2006, IEEE Symposium on*, S. 61–69. IEEE (2006).

[14] V. Havran et al. *Statistical comparison of ray-shooting efficiency schemes* (2000).

[15] A. Knoll. *A Survey of Octree Volume Rendering Methods* (2006).

[16] J. D. MacDonald et al. *Heuristics for ray tracing using space subdivision.* The Visual Computer, Bd. 6, Nr. 3:153–166 (1990).

[17] H. Coxeter. *Introduction to geometry* (1969).

[18] W. Dahmen et al. *Numerik für Ingenieure und Naturwissenschaftler*. Springer (2008).

[19] M. Brill. *Computergrafik - Raytracing: Schnittpunktberechnungen* (2011).

[20] W. Purgathofer. *Computergrafik I - Raytracing* (2009).

[21] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 2.2*. High Performance Computing Center Stuttgart (HLRS) (2009).

[22] P. Rechenberg. *Informatik-Handbuch*. Hanser Verlag (2006).

[23] H. Samet. *Implementing ray tracing with octrees and neighbor finding.* Computers & Graphics, Bd. 13, Nr. 4:445–460 (1989).

[24] V. Havran. *A summary of octree ray traversal algorithms*. Ray Tracing News, Bd. 12, Nr. 2:11–23 (1999).

[25] J. Revelles et al. *An Efficient Parametric Algorithm for Octree Traversal.* In *WSCG* (2000).

[26] A. Glassner. *Space subdivision for fast ray tracing*. IEEE Computer Graphics and applications, Bd. 4:15–22 (1984).

Jül-4360
Januar 2013
ISSN 0944-2952